Introduction to

Antoine Prouvost

20th of March 2020

Supervised Learning

$$\min_{\theta} \quad \mathbb{E}\left[\ell(f_\theta(X), Y)\right]$$

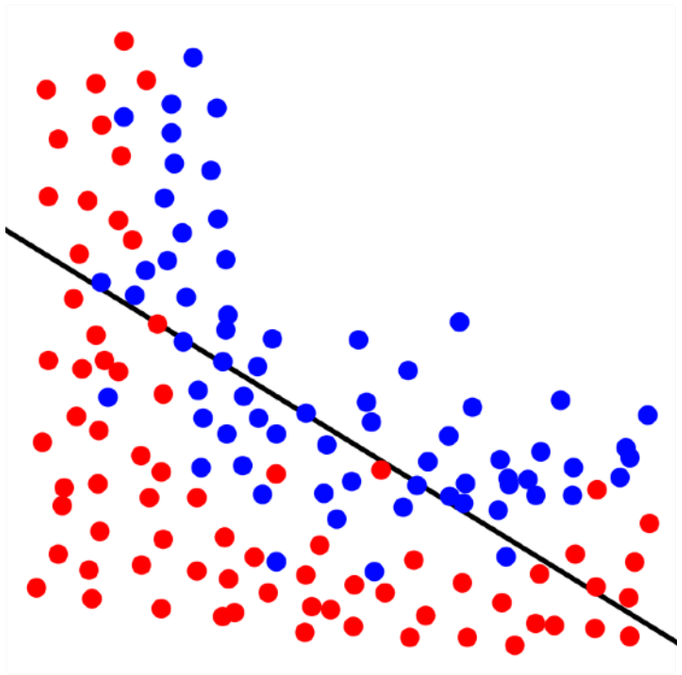
- Expectation over unseen examples
- ℓ is the loss function
- f_θ is the predictor (neural network) with parameters θ
- (X, Y) are random input/target variables

Training objective

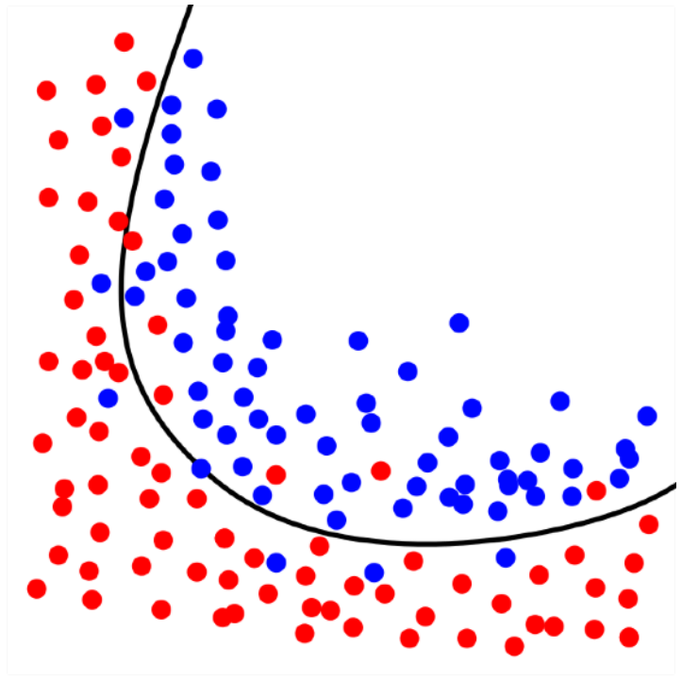
$$\min_{\theta} \quad \mathscr{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(f_\theta(x_i), y_i)$$

- N examples in the training set
- ℓ is the loss function (problem specific, *e.g.* MSE)
- f_θ is the predictor (neural network) with parameters θ
- (x_i, y_i) are the training examples

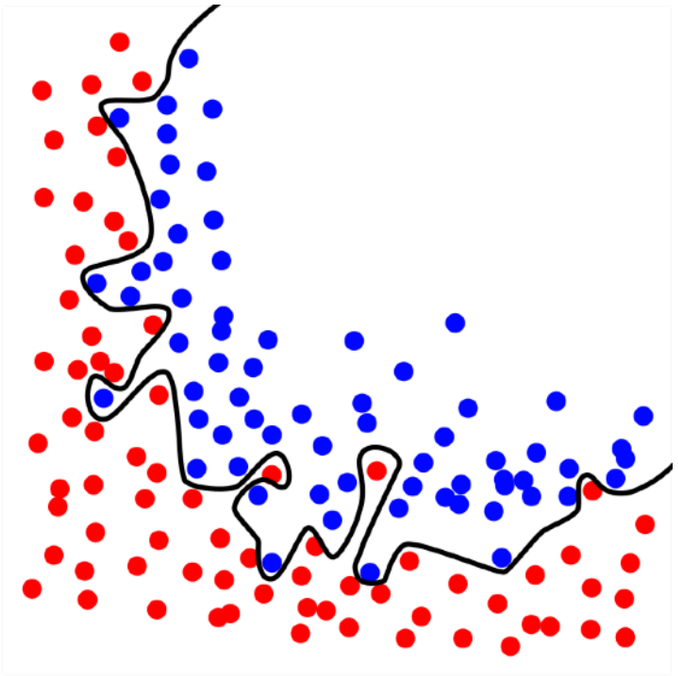
Goodness of Fit



Underfit

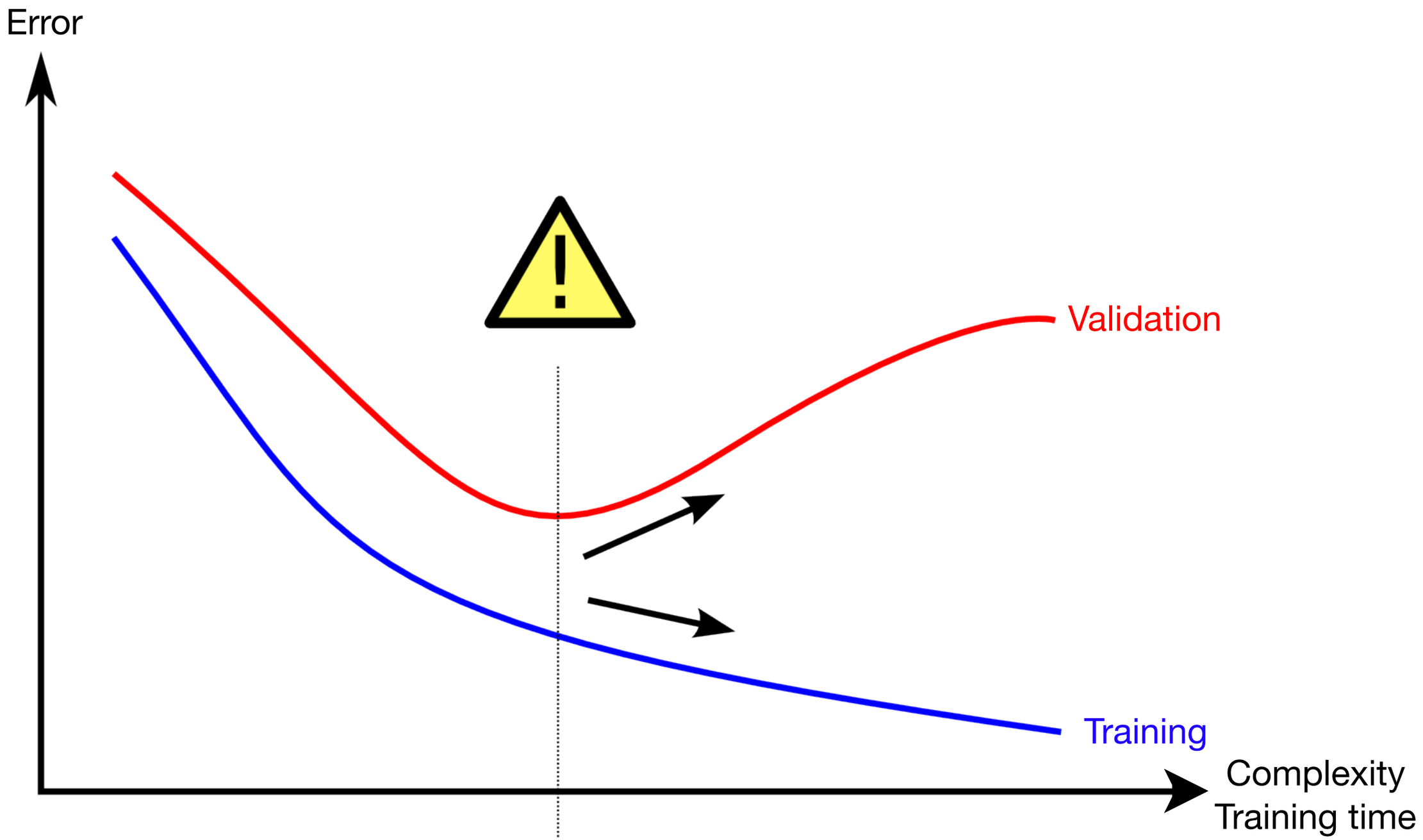


Good fit

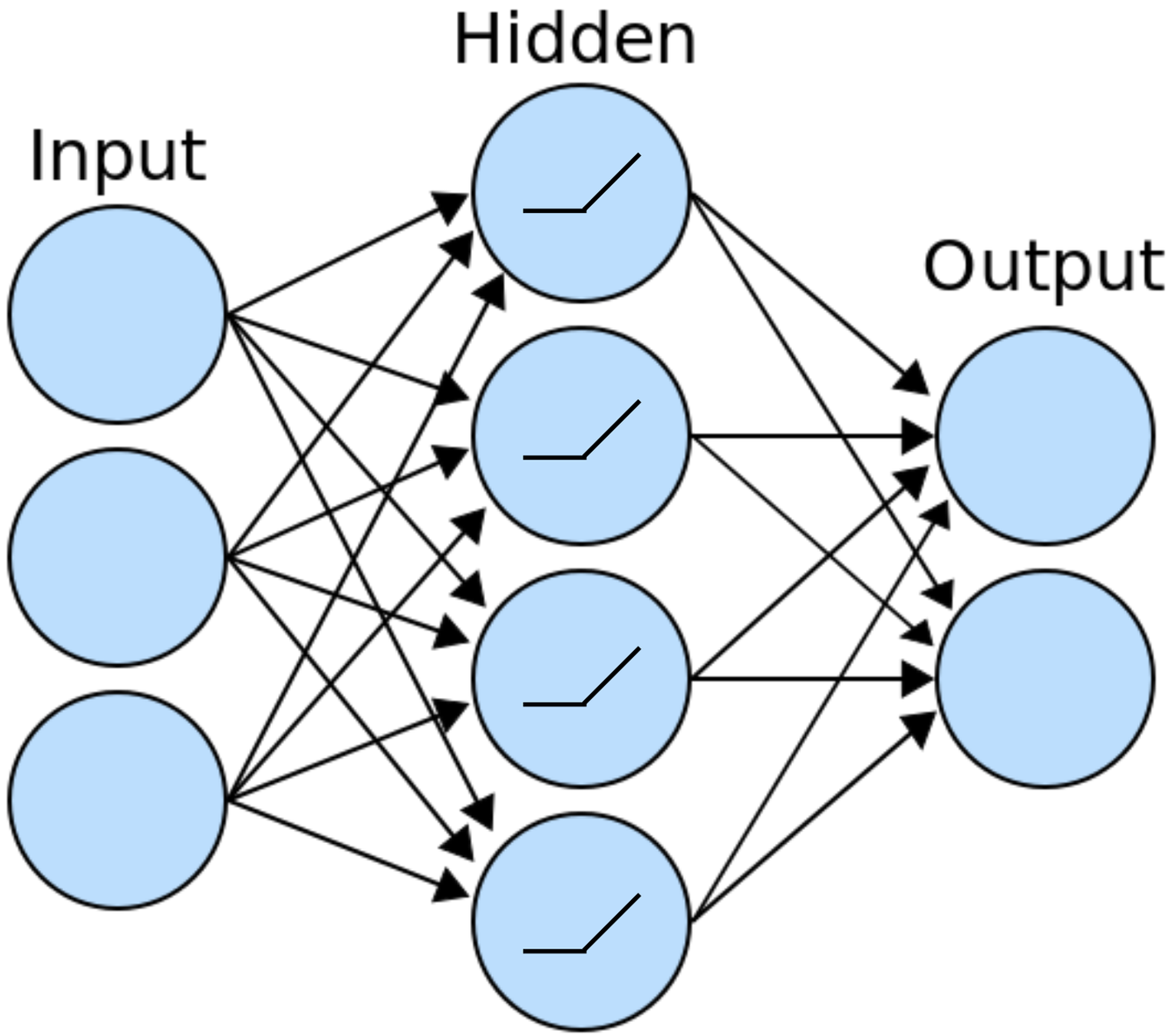


Overfit

Goodness of Fit



Neural Network



Data Layout

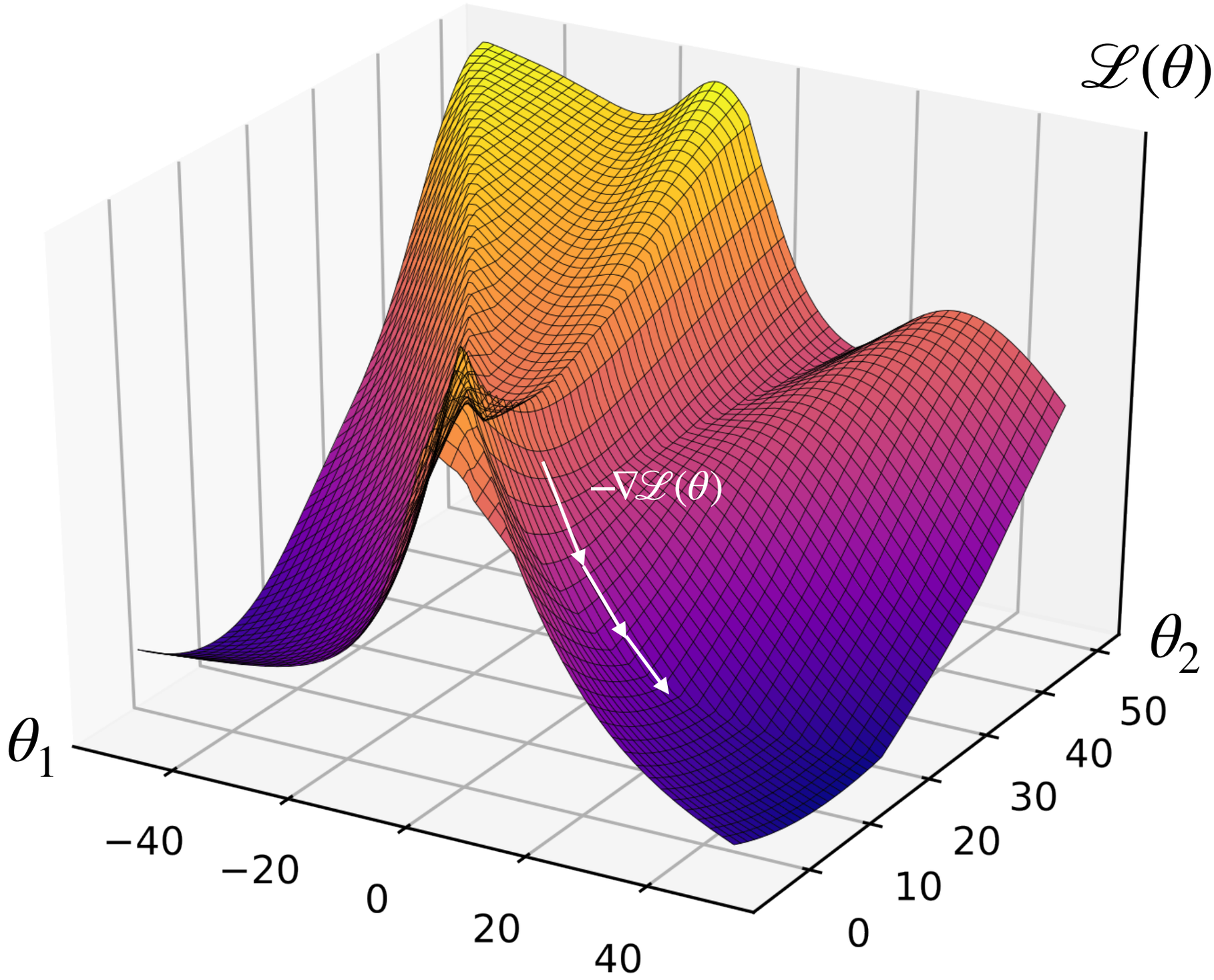
kth feature

ith example

```
tensor([[0.6376, 0.9057, 0.6846, 0.7117, 0.8487, 0.0744, 0.9518],
        [0.2923, 0.7165, 0.1973, 0.2397, 0.5771, 0.5638, 0.9993],
        [0.2369, 0.9512, 0.7222, 0.2426, 0.1483, 0.6278, 0.7791],
        [0.2237, 0.5489, 0.9370, 0.8457, 0.9986, 0.7233, 0.7889],
        [0.8468, 0.2340, 0.0911, 0.4262, 0.1687, 0.6305, 0.7508],
        [0.3378, 0.1670, 0.3443, 0.0444, 0.2323, 0.8563, 0.0130],
        [0.5080, 0.7051, 0.8770, 0.3361, 0.6279, 0.9214, 0.7670],
        [0.5072, 0.1404, 0.3881, 0.9701, 0.6215, 0.1271, 0.5980],
        [0.6257, 0.4641, 0.3438, 0.8921, 0.8798, 0.0657, 0.1461],
        [0.9679, 0.6459, 0.8980, 0.2363, 0.9688, 0.4283, 0.2100],
        [0.4895, 0.4566, 0.3123, 0.2635, 0.5453, 0.0066, 0.5287],
        [0.1535, 0.0807, 0.5558, 0.9945, 0.2332, 0.8090, 0.1357],
        [0.7637, 0.5319, 0.8635, 0.8072, 0.5585, 0.1226, 0.3948],
        [0.5267, 0.1512, 0.0959, 0.0583, 0.1057, 0.5622, 0.5102],
        [0.3715, 0.4426, 0.4319, 0.9708, 0.4003, 0.6172, 0.8253],
        [0.5909, 0.8879, 0.1540, 0.9327, 0.9577, 0.8344, 0.8575],
        [0.4352, 0.2752, 0.2222, 0.6507, 0.8302, 0.9882, 0.1323],
        [0.3817, 0.9683, 0.0256, 0.8480, 0.9195, 0.5951, 0.1267],
        [0.9772, 0.1110, 0.4525, 0.2388, 0.3032, 0.4890, 0.2888],
        [0.1775, 0.7266, 0.9099, 0.8389, 0.5672, 0.2913, 0.8584],
        [0.2952, 0.0455, 0.5989, 0.9106, 0.6178, 0.6184, 0.6094],
        [0.2643, 0.0340, 0.7447, 0.2637, 0.2932, 0.7452, 0.6972],
        [0.7966, 0.0639, 0.8146, 0.7864, 0.0794, 0.7675, 0.1843],
        [0.7056, 0.3377, 0.9615, 0.6633, 0.0111, 0.3464, 0.8024],
        [0.7954, 0.2537, 0.7292, 0.9774, 0.9494, 0.5149, 0.1547],
        [0.0217, 0.0160, 0.0199, 0.2547, 0.5276, 0.8306, 0.3736],
        [0.2487, 0.7770, 0.7682, 0.0269, 0.9355, 0.1020, 0.3830],
        [0.8083, 0.5732, 0.6287, 0.1231, 0.1675, 0.4726, 0.3766],
        [0.1188, 0.6217, 0.7025, 0.4512, 0.1456, 0.7578, 0.8339],
        [0.1987, 0.6446, 0.3461, 0.8246, 0.7431, 0.9381, 0.0073],
        [0.2910, 0.9862, 0.8221, 0.2347, 0.2430, 0.3811, 0.2865],
        [0.3573, 0.9250, 0.0106, 0.6855, 0.7090, 0.5050, 0.6747],
        [0.3651, 0.5485, 0.5214, 0.4207, 0.1140, 0.3645, 0.1332],
        [0.4607, 0.0735, 0.3656, 0.2844, 0.7690, 0.8687, 0.8679],
```

Data Layout

Gradient Descent



Gradient Descent

$$\theta \leftarrow \theta - \lambda \nabla \mathscr{L}(\theta)$$

- θ parameters of the models (weights)
- $\nabla \mathscr{L}(\theta)$ gradient of the loss with regard to the parameters
- λ learning rate

Stochastic Gradient

```
tensor([[0.6376, 0.9057, 0.6846, 0.7117, 0.8487, 0.0744, 0.9518],
        [0.2923, 0.7165, 0.1973, 0.2397, 0.5771, 0.5638, 0.9993],
        [0.2369, 0.9512, 0.7222, 0.2426, 0.1483, 0.6278, 0.7791],
        [0.2237, 0.5489, 0.9370, 0.8457, 0.9986, 0.7233, 0.7889],
        [0.8468, 0.2340, 0.0911, 0.4262, 0.1687, 0.6305, 0.7508],
        [0.3378, 0.1670, 0.3443, 0.0444, 0.2323, 0.8563, 0.0130],
        [0.5080, 0.7051, 0.8770, 0.3361, 0.6279, 0.9214, 0.7670],
        [0.5072, 0.1404, 0.3881, 0.9701, 0.6215, 0.1271, 0.5980],
        [0.6257, 0.4641, 0.3438, 0.8921, 0.8798, 0.0657, 0.1461],
        [0.9679, 0.6459, 0.8980, 0.2363, 0.9688, 0.4283, 0.2100],
        [0.4895, 0.4566, 0.3123, 0.2635, 0.5453, 0.0066, 0.5287],
        [0.1535, 0.0807, 0.5558, 0.9945, 0.2332, 0.8090, 0.1357],
        [0.7637, 0.5319, 0.8635, 0.8072, 0.5585, 0.1226, 0.3948],
        [0.5267, 0.1512, 0.0959, 0.0583, 0.1057, 0.5622, 0.5102],
        [0.3715, 0.4426, 0.4319, 0.9708, 0.4003, 0.6172, 0.8253],
        [0.5909, 0.8879, 0.1540, 0.9327, 0.9577, 0.8344, 0.8575],
        [0.4352, 0.2752, 0.2222, 0.6507, 0.8302, 0.9882, 0.1323],
        [0.3817, 0.9683, 0.0256, 0.8480, 0.9195, 0.5951, 0.1267],
        [0.9772, 0.1110, 0.4525, 0.2388, 0.3032, 0.4890, 0.2888],
        [0.1775, 0.7266, 0.9099, 0.8389, 0.5672, 0.2913, 0.8584],
        [0.2952, 0.0455, 0.5989, 0.9106, 0.6178, 0.6184, 0.6094],
        [0.2643, 0.0340, 0.7447, 0.2637, 0.2932, 0.7452, 0.6972],
        [0.7966, 0.0639, 0.8146, 0.7864, 0.0794, 0.7675, 0.1843],
        [0.7056, 0.3377, 0.9615, 0.6633, 0.0111, 0.3464, 0.8024],
        [0.7954, 0.2537, 0.7292, 0.9774, 0.9494, 0.5149, 0.1547],
        [0.0217, 0.0160, 0.0199, 0.2547, 0.5276, 0.8306, 0.3736],
        [0.2487, 0.7770, 0.7682, 0.0269, 0.9355, 0.1020, 0.3830],
        [0.8083, 0.5732, 0.6287, 0.1231, 0.1675, 0.4726, 0.3766],
        [0.1188, 0.6217, 0.7025, 0.4512, 0.1456, 0.7578, 0.8339],
        [0.1987, 0.6446, 0.3461, 0.8246, 0.7431, 0.9381, 0.0073],
        [0.2910, 0.9862, 0.8221, 0.2347, 0.2430, 0.3811, 0.2865],
        [0.3573, 0.9250, 0.0106, 0.6855, 0.7090, 0.5050, 0.6747],
        [0.3651, 0.5485, 0.5214, 0.4207, 0.1140, 0.3645, 0.1332],
        [0.4607, 0.0735, 0.3656, 0.2844, 0.7690, 0.8687, 0.8679],
```

Examples used to compute average loss on 2nd mini-batch

Batch size

Stochastic Gradient

```
tensor([[0.6376, 0.9057, 0.6846, 0.7117, 0.8487, 0.0744, 0.9518],
        [0.2923, 0.7165, 0.1973, 0.2397, 0.5771, 0.5638, 0.9993],
        [0.2369, 0.9512, 0.7222, 0.2426, 0.1483, 0.6278, 0.7791],
        [0.2237, 0.5489, 0.9370, 0.8457, 0.9986, 0.7233, 0.7889],
        [0.8468, 0.2340, 0.0911, 0.4262, 0.1687, 0.6305, 0.7508],
        [0.3378, 0.1670, 0.3443, 0.0444, 0.2323, 0.8563, 0.0130],
        [0.5080, 0.7051, 0.8770, 0.3361, 0.6279, 0.9214, 0.7670],
        [0.5072, 0.1404, 0.3881, 0.9701, 0.6215, 0.1271, 0.5980],
        [0.6257, 0.4641, 0.3438, 0.8921, 0.8798, 0.0657, 0.1461],
        [0.9679, 0.6459, 0.8980, 0.2363, 0.9688, 0.4283, 0.2100],
        [0.4895, 0.4566, 0.3123, 0.2635, 0.5453, 0.0066, 0.5287],
        [0.1535, 0.0807, 0.5558, 0.9945, 0.2332, 0.8090, 0.1357],
        [0.7637, 0.5319, 0.8635, 0.8072, 0.5585, 0.1226, 0.3948],
        [0.5267, 0.1512, 0.0959, 0.0583, 0.1057, 0.5622, 0.5102],
        [0.3715, 0.4426, 0.4319, 0.9708, 0.4003, 0.6172, 0.8253],
        [0.5909, 0.8879, 0.1540, 0.9327, 0.9577, 0.8344, 0.8575],
        [0.4352, 0.2752, 0.2222, 0.6507, 0.8302, 0.9882, 0.1323],
        [0.3817, 0.9683, 0.0256, 0.8480, 0.9195, 0.5951, 0.1267],
        [0.9772, 0.1110, 0.4525, 0.2388, 0.3032, 0.4890, 0.2888],
        [0.1775, 0.7266, 0.9099, 0.8389, 0.5672, 0.2913, 0.8584],
        [0.2952, 0.0455, 0.5989, 0.9106, 0.6178, 0.6184, 0.6094],
        [0.2643, 0.0340, 0.7447, 0.2637, 0.2932, 0.7452, 0.6972],
        [0.7966, 0.0639, 0.8146, 0.7864, 0.0794, 0.7675, 0.1843],
        [0.7056, 0.3377, 0.9615, 0.6633, 0.0111, 0.3464, 0.8024],
        [0.7954, 0.2537, 0.7292, 0.9774, 0.9494, 0.5149, 0.1547],
        [0.0217, 0.0160, 0.0199, 0.2547, 0.5276, 0.8306, 0.3736],
        [0.2487, 0.7770, 0.7682, 0.0269, 0.9355, 0.1020, 0.3830],
        [0.8083, 0.5732, 0.6287, 0.1231, 0.1675, 0.4726, 0.3766],
        [0.1188, 0.6217, 0.7025, 0.4512, 0.1456, 0.7578, 0.8339],
        [0.1987, 0.6446, 0.3461, 0.8246, 0.7431, 0.9381, 0.0073],
        [0.2910, 0.9862, 0.8221, 0.2347, 0.2430, 0.3811, 0.2865],
        [0.3573, 0.9250, 0.0106, 0.6855, 0.7090, 0.5050, 0.6747],
        [0.3651, 0.5485, 0.5214, 0.4207, 0.1140, 0.3645, 0.1332],
        [0.4607, 0.0735, 0.3656, 0.2844, 0.7690, 0.8687, 0.8679],
```

Random draw
Without replacement

Chain Rule

$$h : x \mapsto g(f(x))$$

$$h' : x \mapsto g'(f(x)) \times f'(x)$$

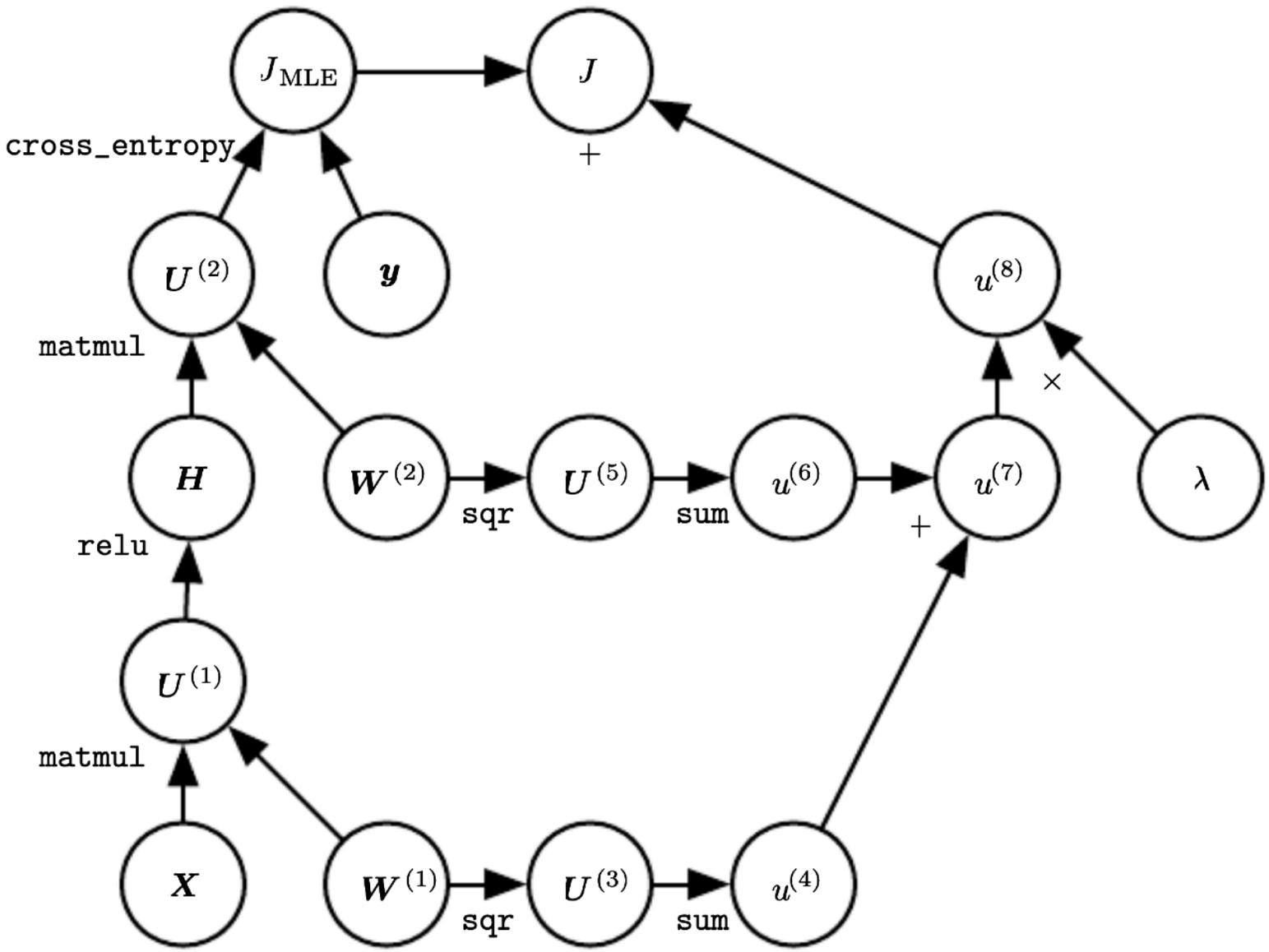
$$y = f(x) \qquad z = g(f(x))$$

$$\frac{\mathrm{d}z}{\mathrm{d}x} = \frac{\mathrm{d}z}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}x}$$

Chain Rule

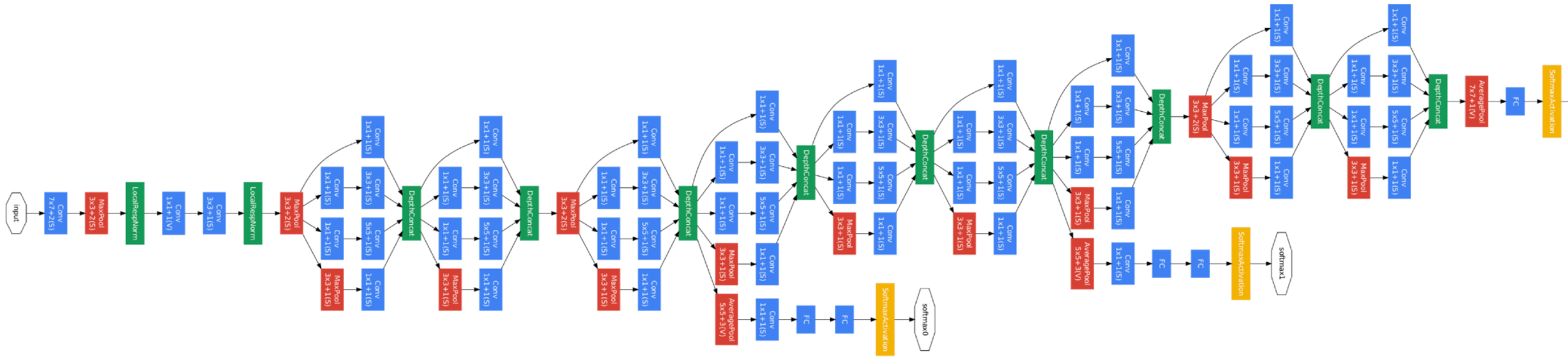
$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

Compute Graph



Compute graph for a MLP (Goodfellow et al.)

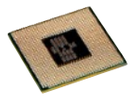
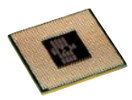
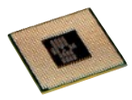
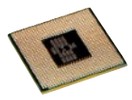
Compute Graph in Practice



GoogLeNet (2015)

Graphical Processing Unit

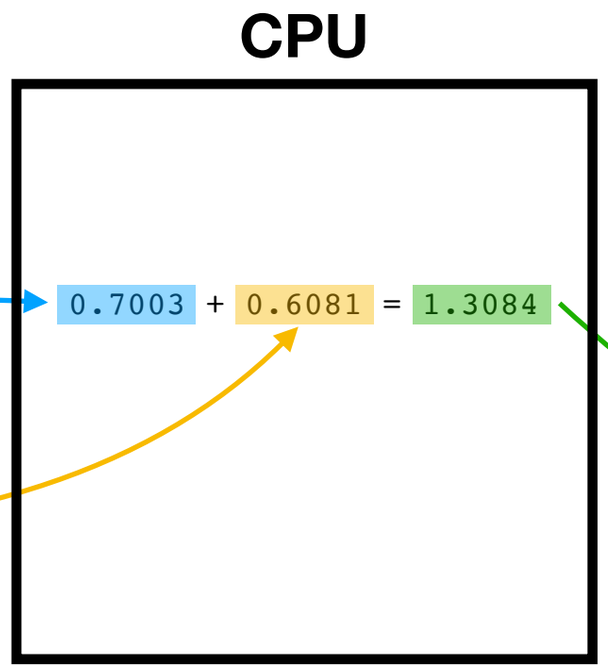
GPU vs CPU

CPU	GPU
Few computation units 	 Numerous computation units
Low latency 	 High throughput
Run complex code 	 Run simple parallel code
Good for serial processing 	 Good for parallel processing

Vector addition on CPU

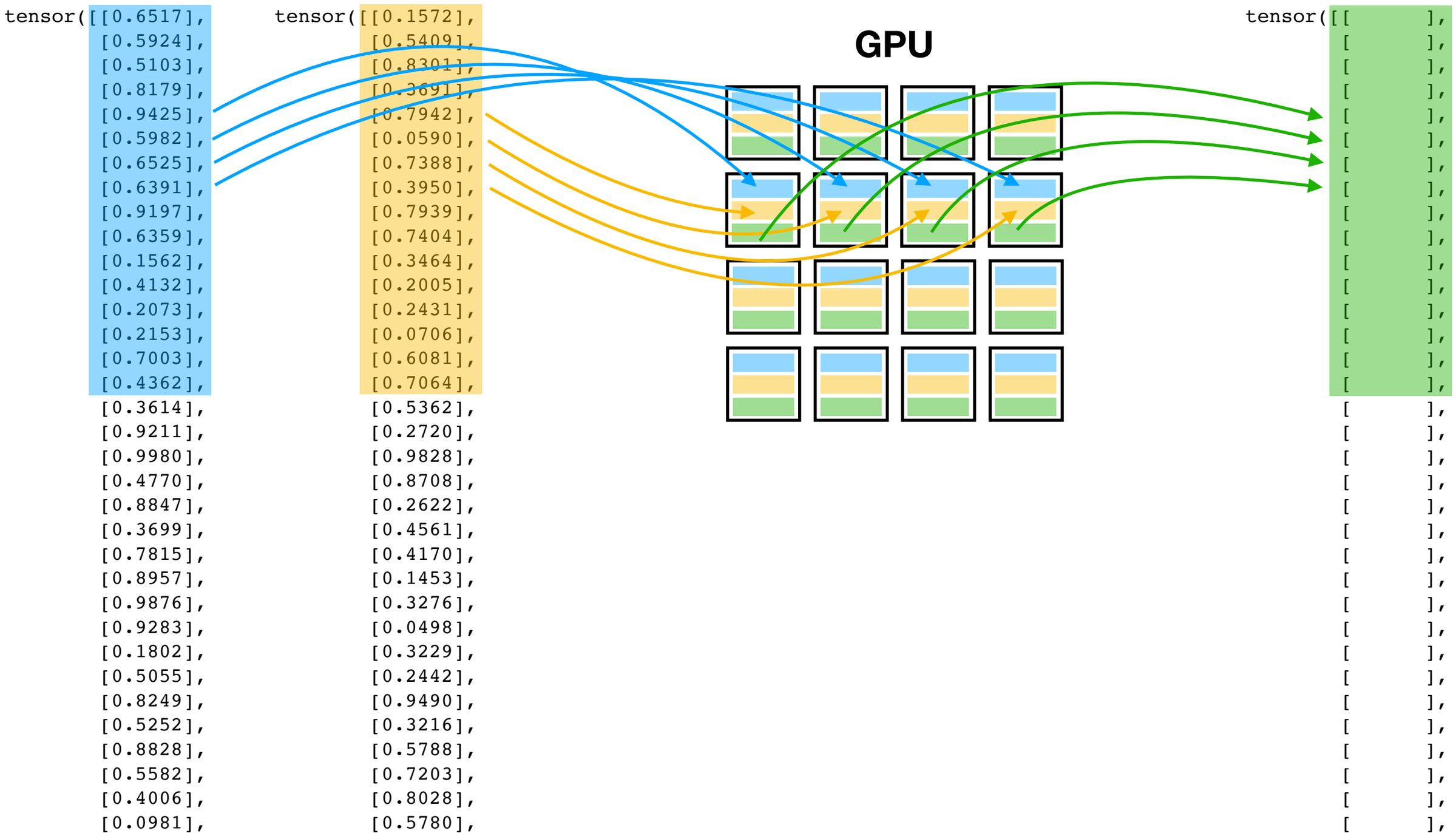
```
tensor([[0.6517],
        [0.5924],
        [0.5103],
        [0.8179],
        [0.9425],
        [0.5982],
        [0.6525],
        [0.6391],
        [0.9197],
        [0.6359],
        [0.1562],
        [0.4132],
        [0.2073],
        [0.2153],
        [0.7003],
        [0.4362],
        [0.3614],
        [0.9211],
        [0.9980],
        [0.4770],
        [0.8847],
        [0.3699],
        [0.7815],
        [0.8957],
        [0.9876],
        [0.9283],
        [0.1802],
        [0.5055],
        [0.8249],
        [0.5252],
        [0.8828],
        [0.5582],
        [0.4006],
        [0.0981],
```

```
tensor([[0.1572],
        [0.5409],
        [0.8301],
        [0.3691],
        [0.7942],
        [0.0590],
        [0.7388],
        [0.3950],
        [0.7939],
        [0.7404],
        [0.3464],
        [0.2005],
        [0.2431],
        [0.0706],
        [0.6081],
        [0.7064],
        [0.5362],
        [0.2720],
        [0.9828],
        [0.8708],
        [0.2622],
        [0.4561],
        [0.4170],
        [0.1453],
        [0.3276],
        [0.0498],
        [0.3229],
        [0.2442],
        [0.9490],
        [0.3216],
        [0.5788],
        [0.7203],
        [0.8028],
        [0.5780],
```

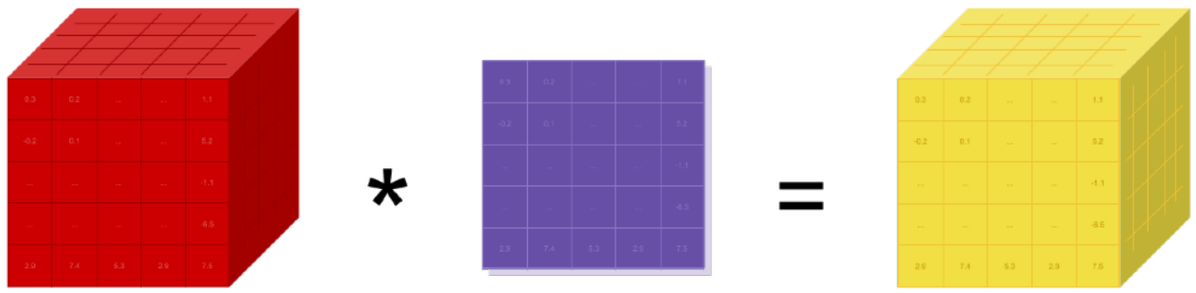



```
tensor([[0.8089],
        [1.1333],
        [1.3405],
        [1.1870],
        [1.7367],
        [0.6573],
        [1.3914],
        [1.0341],
        [1.7136],
        [1.3763],
        [0.5026],
        [0.6137],
        [0.4504],
        [0.2859],
        [      ],
        [      ],
        [      ],
        [      ],
        [      ],
        [      ],
        [      ],
        [      ],
        [      ],
        [      ],
        [      ],
        [      ],
        [      ],
        [      ],
        [      ],
        [      ],
        [      ],
        [      ],
        [      ],
        [      ],
```

Vector addition on GPU



PyTorch

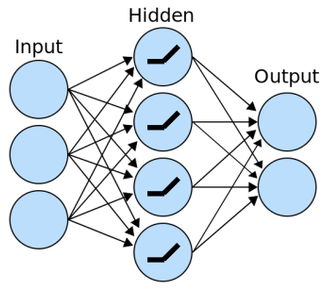


+

+

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

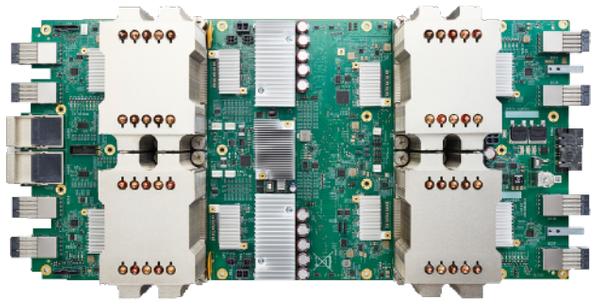
+



PyTorch In The Wild

ONNX

Google Cloud Platform



colab

amazon web services™

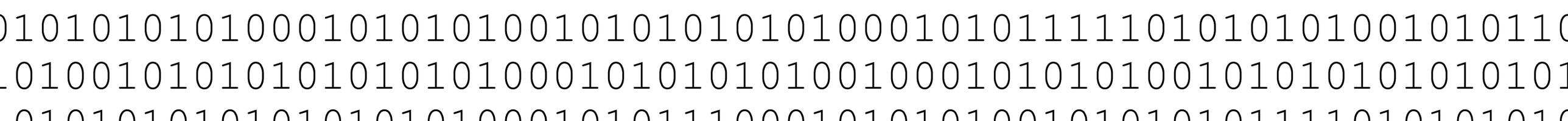
Azure

TorchScript

`torch.jit`

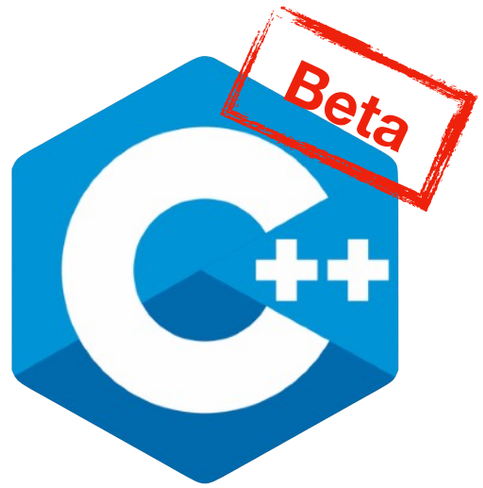
Dynamic execution graph ⟶ Static execution graph

- Compile PyTorch code;
- Improved performances on fine-grained operations;
- Export models to use in other frontends.



Frontends

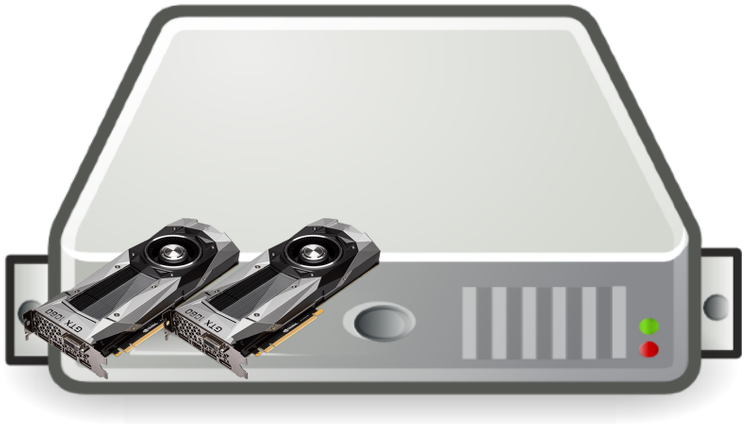
Full API

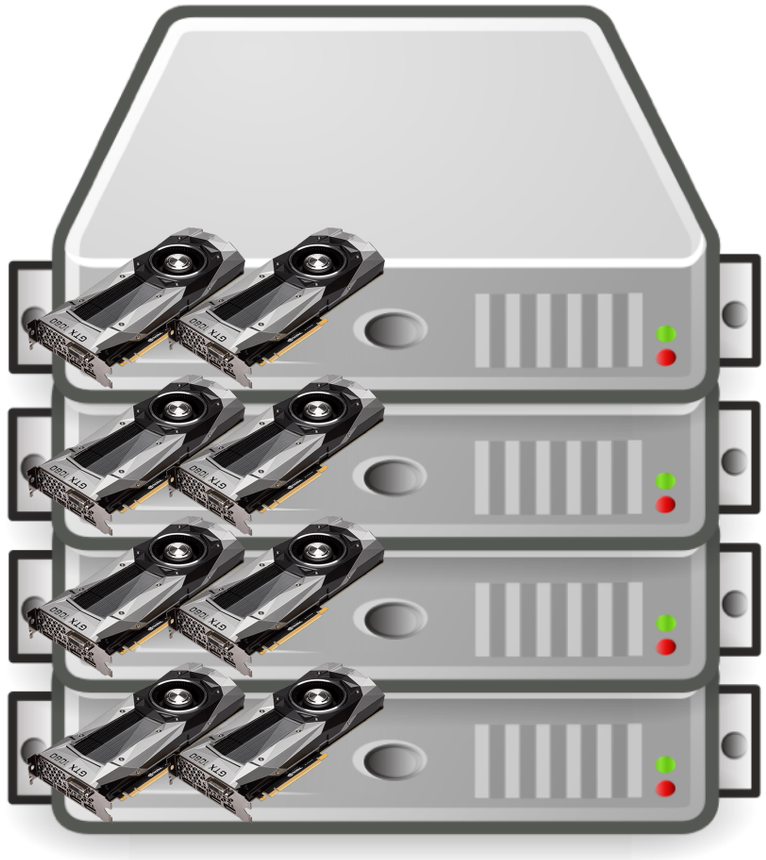


Active development

Currently inference/serving only

Distributed

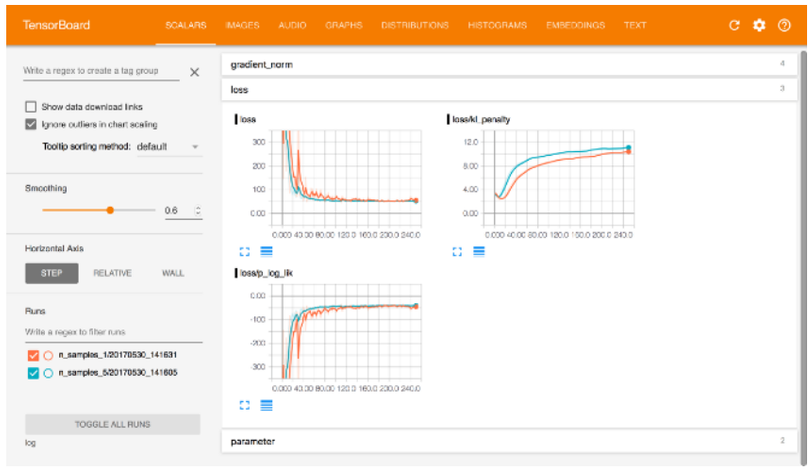




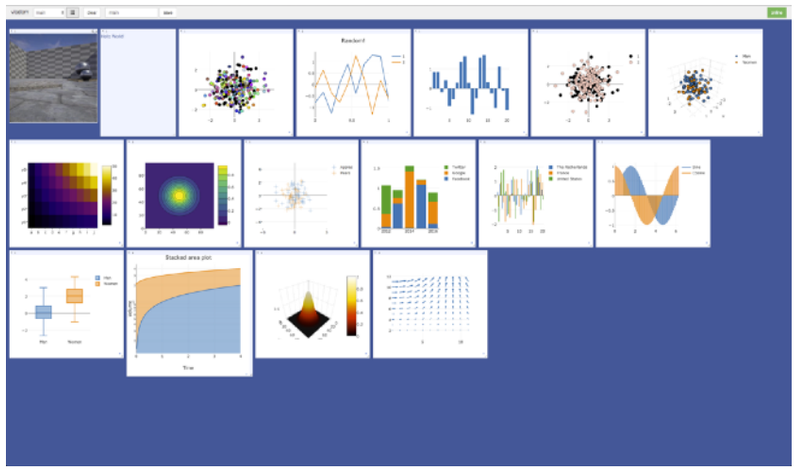
torch.multiprocessing

torch.distributed

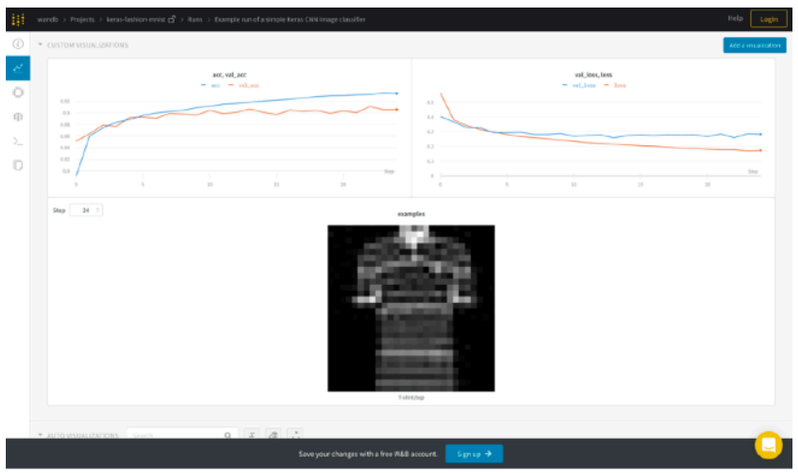
Visualization



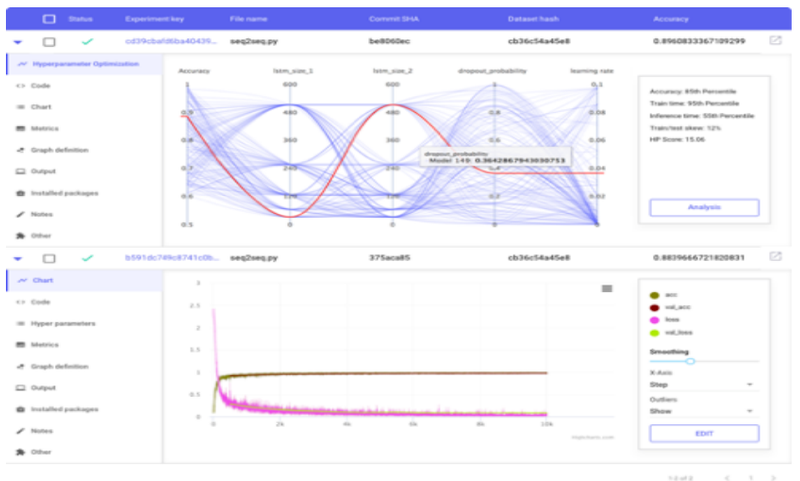
Tensorboard



Visdom



wandb.ai (service)



Comet.ml (service)

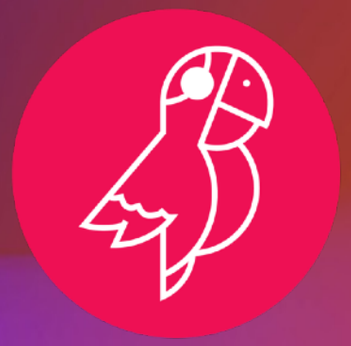
adversarial torch

BoTorch

CrypTen

flair

DeepGraphLibrary



SKORCH

fast.ai

Captum

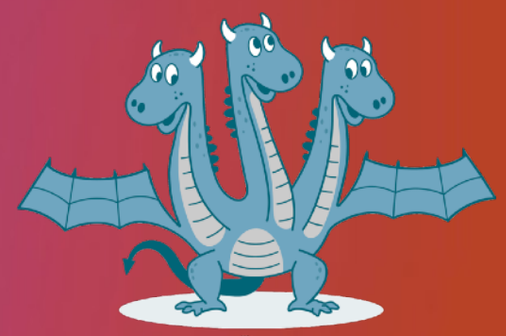
Catalyst

AllenNLP

XANADU

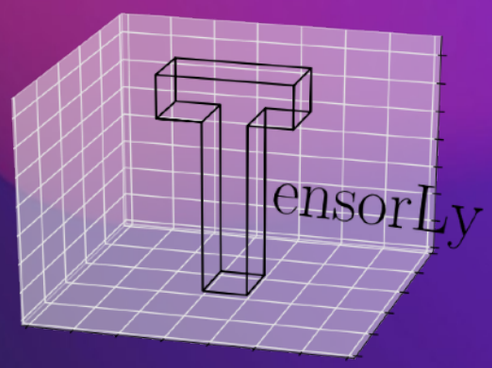
Ignite

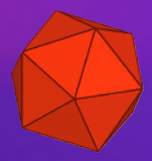
GLOW



ONNX

kornia





PyTorch geometric

Torchbearer

Poutyne

And many more…

PyTorch or TensorFlow ?

Both have great ecosystems and can implement all models.

- Dynamic graph by design
- Great streamlined user interface
- Easier to use / debug

- Static graph by designed
- Messier API
- Better Tensorboard integration
- Easier to put in production
- TF Lite / TF Hub / TF.js

But TF is getting eager and PyTorch C++ frontend / JIT
They are converging