Deep learning with PyTorch

Luca Antiga, Orobix

Deep learning with PyTorch

- Neural networks and back-propagation
- **PyTorch basics**
- Deep learning with PyTorch
- Deep learning for g2net data

PyTorch basics

Luca Antiga, Orobix

This slide deck contains:

- Figures from Deep learning with PyTorch, FB AI ed
  `https://pytorch.org/deep-learning-with-pytorch`
- Slides from Gilles Louppe's course
  `https://github.com/glouppe/info8010-deep-learning`
- Figures from Sebastian Raschka's repo
  `https://github.com/rasbt/deeplearning-models`

FROM RESEARCH TO PRODUCTION

An open source machine learning framework that accelerates the path from research prototyping to production deployment.

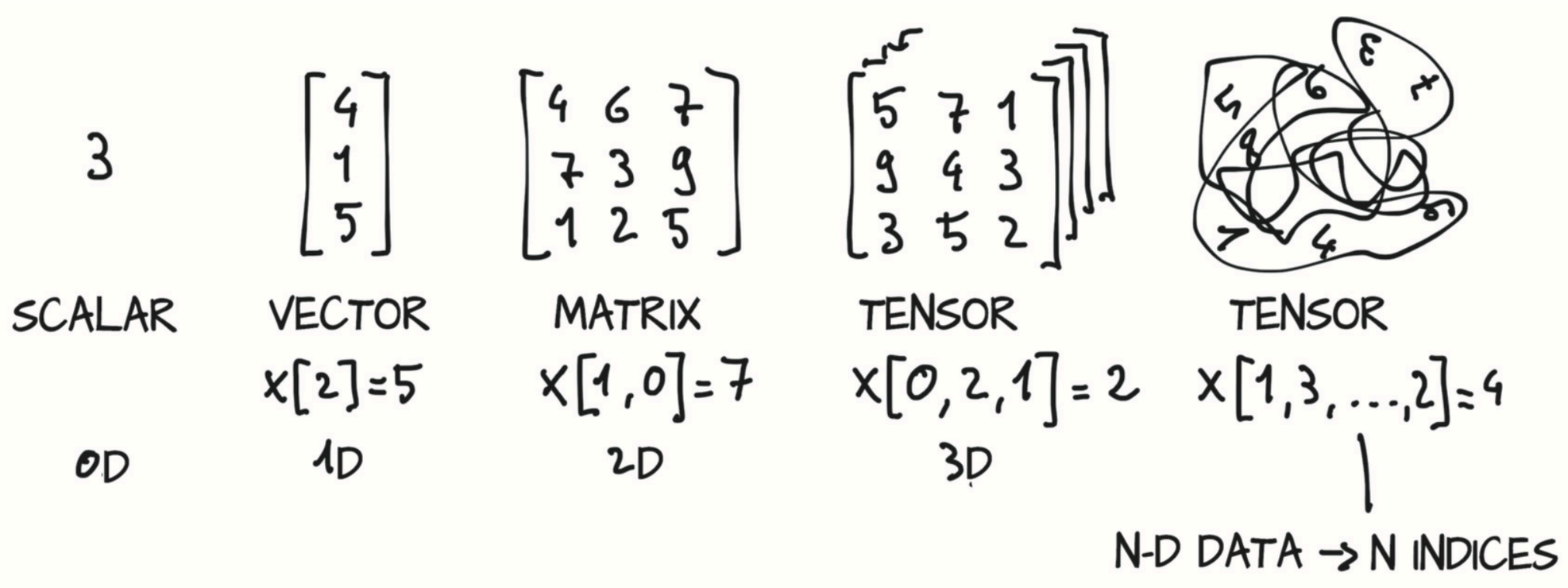
Get Started >

PyTorch 1.4 is now available - adds ability to do fine grain build level customization for PyTorch Mobile, updated domain libraries, and new experimenta...

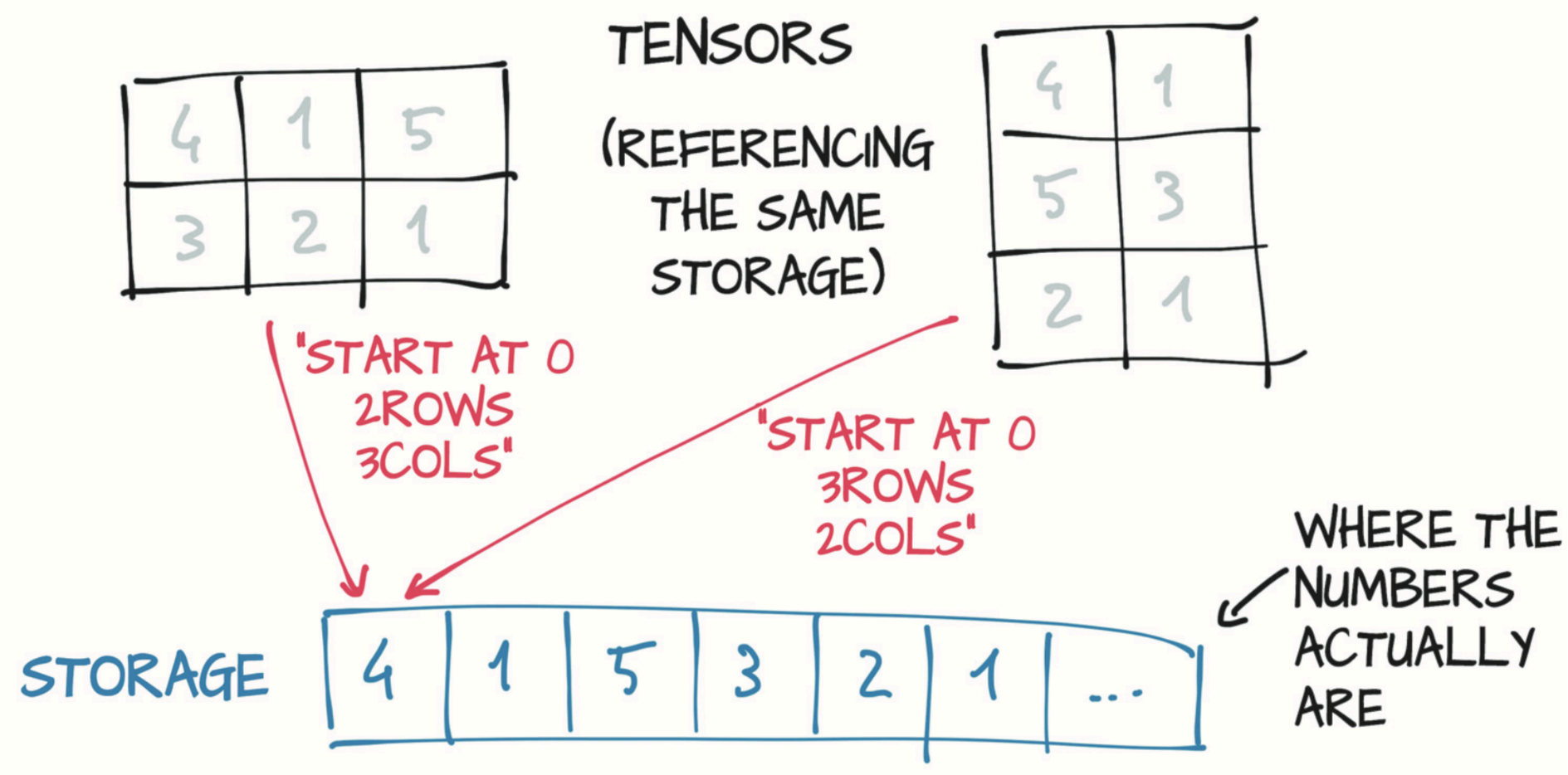
KEY FEATURES & CAPABILITIES

See all Features >

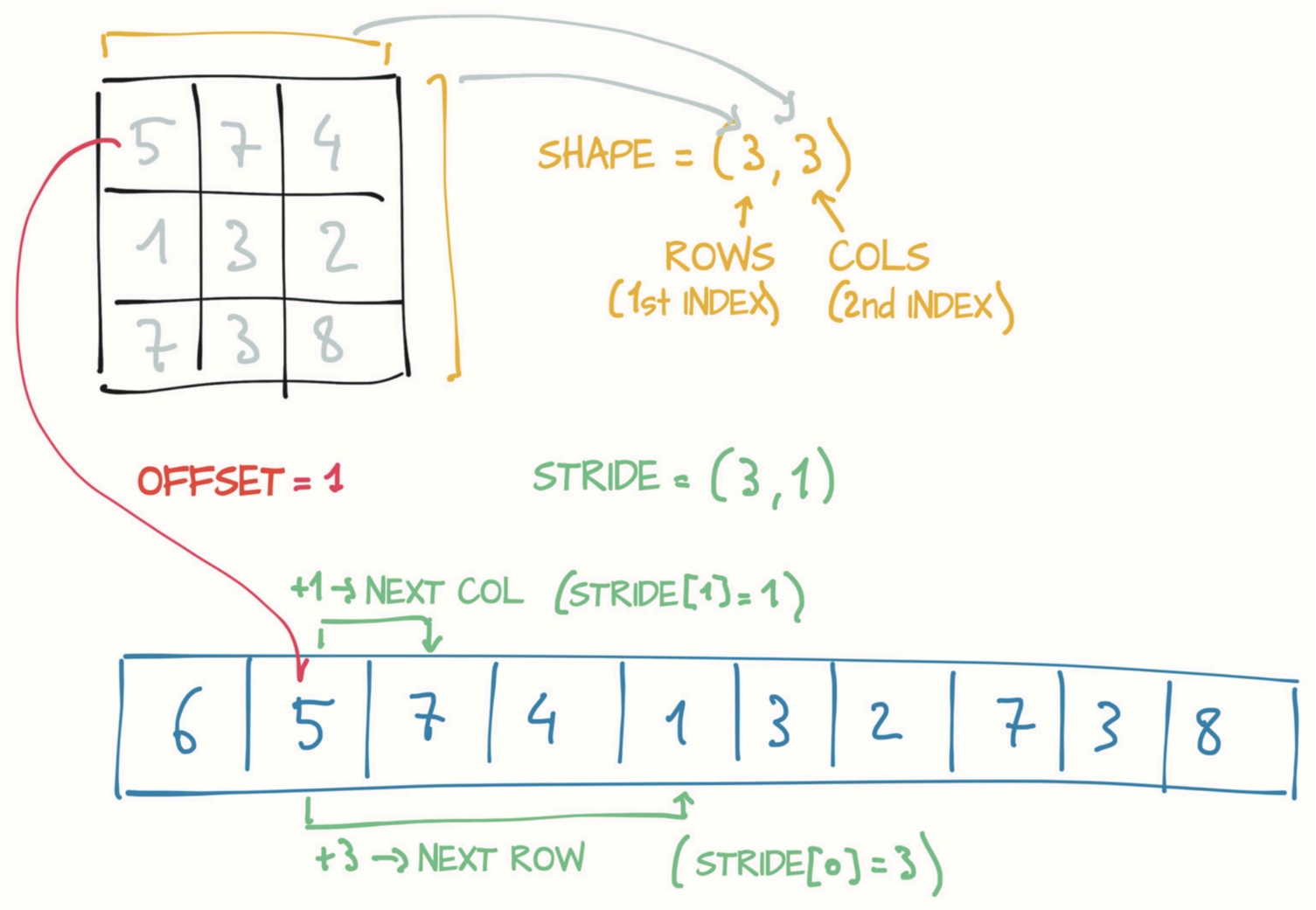
Tensors


3
SCALAR
0D
VECTOR
x[2]=5
1D
MATRIX
x[1,0]=7
2D
TENSOR
x[0,2,1]=2
3D
TENSOR
x[1,3,...,2]=4
N-D DATA → N INDICES

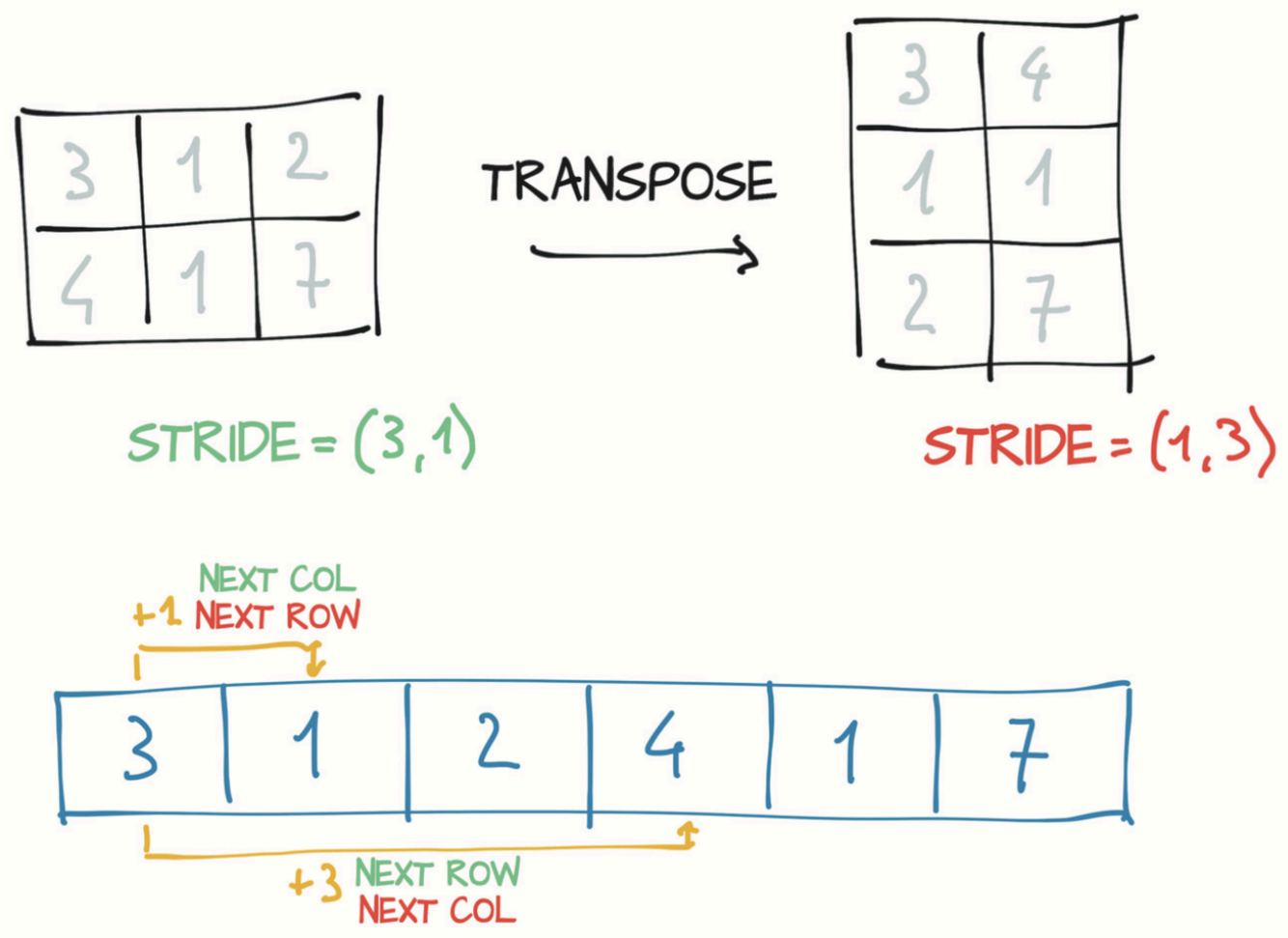
Tensors and storages



Tensor metadata



Transposing



Practice

Open a notebook or go to Colaboratory

https://colab.research.google.com

Automatic differentiation

To minimize $\mathcal{L}(\theta)$ with stochastic gradient descent, we need the gradient $\nabla_\theta \ell(\theta_t)$.

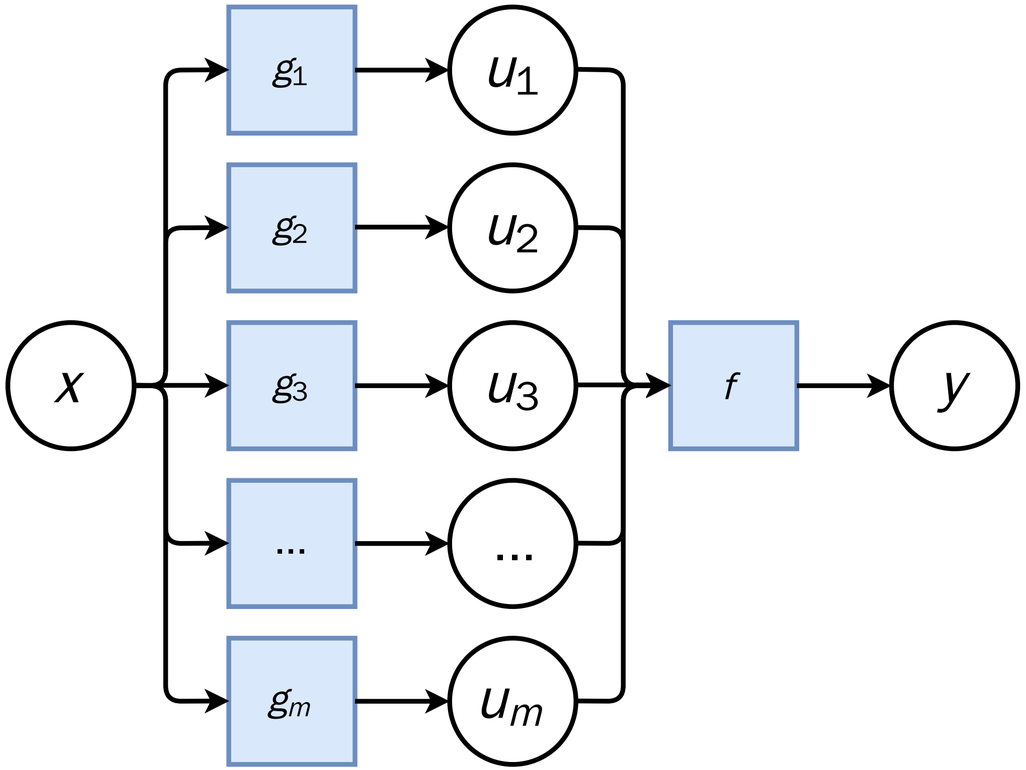
Therefore, we require the evaluation of the (total) derivatives

$$\frac{\mathrm{d}\ell}{\mathrm{d}\mathbf{W}_k}, \frac{\mathrm{d}\ell}{\mathrm{d}\mathbf{b}_k}$$

of the loss ℓ with respect to all model parameters $\mathbf{W}_k, \mathbf{b}_k$, for $k = 1, \ldots, L$.

These derivatives can be evaluated automatically from the computational graph of ℓ using automatic differentiation.

Chain rule



Let us consider a 1-dimensional output composition $f \circ g$, such that

$$y = f(\mathbf{u})$$
$$\mathbf{u} = g(x) = (g_1(x), ..., g_m(x)).$$

Chain rule

The chain rule states that $(f \circ g)' = (f' \circ g)g'$.

For the total derivative, the chain rule generalizes to

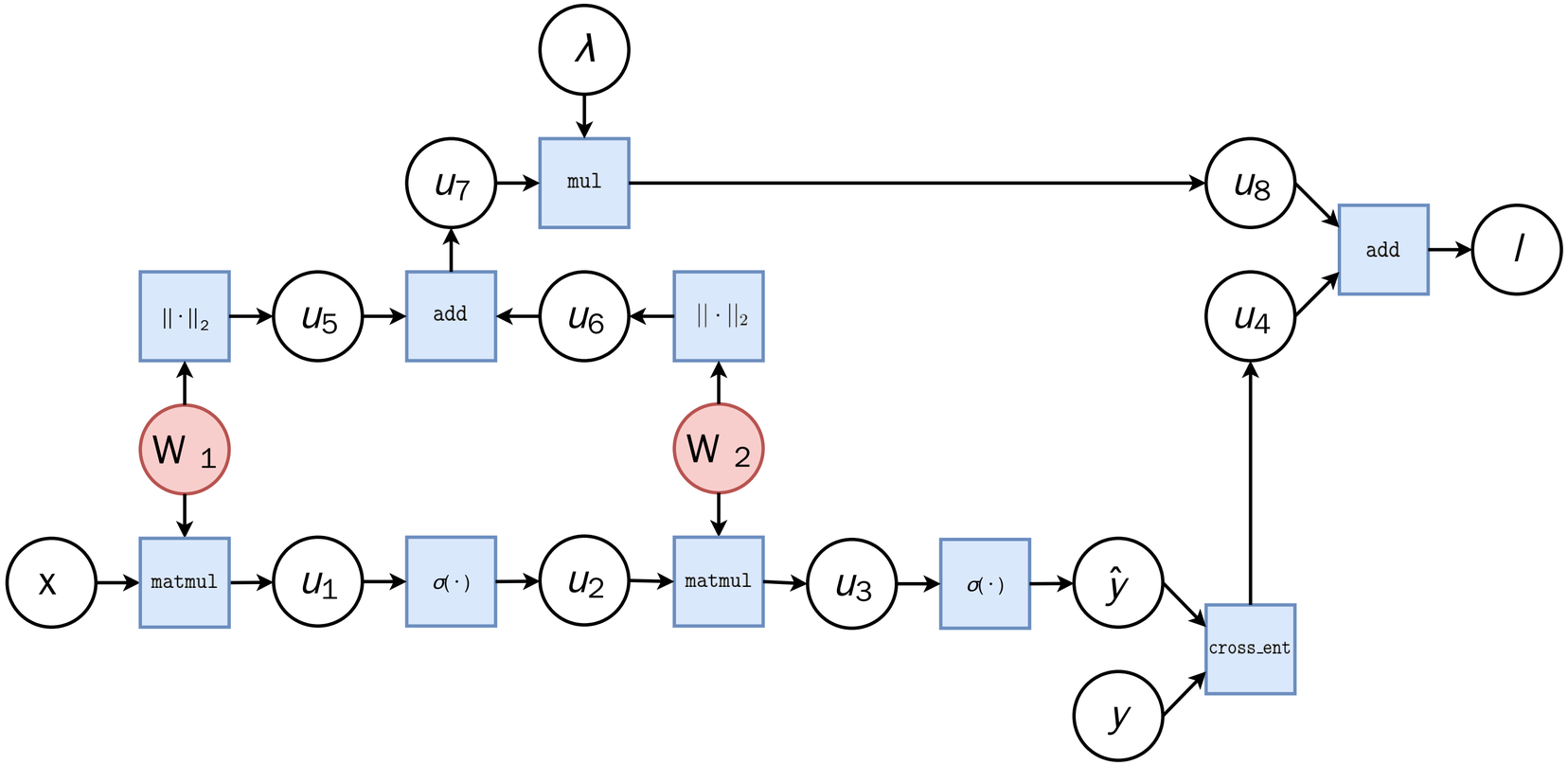
$$\frac{\mathrm{d}y}{\mathrm{d}x} = \sum_{k=1}^{m} \frac{\partial y}{\partial u_k} \underbrace{\frac{\mathrm{d}u_k}{\mathrm{d}x}}_{\text{recursive case}}$$

Reverse automatic differentiation

- Since a neural network is a composition of differentiable functions, the total derivatives of the loss can be evaluated backward, by applying the chain rule recursively over its computational graph.
- The implementation of this procedure is called reverse automatic differentiation.

Forward pass

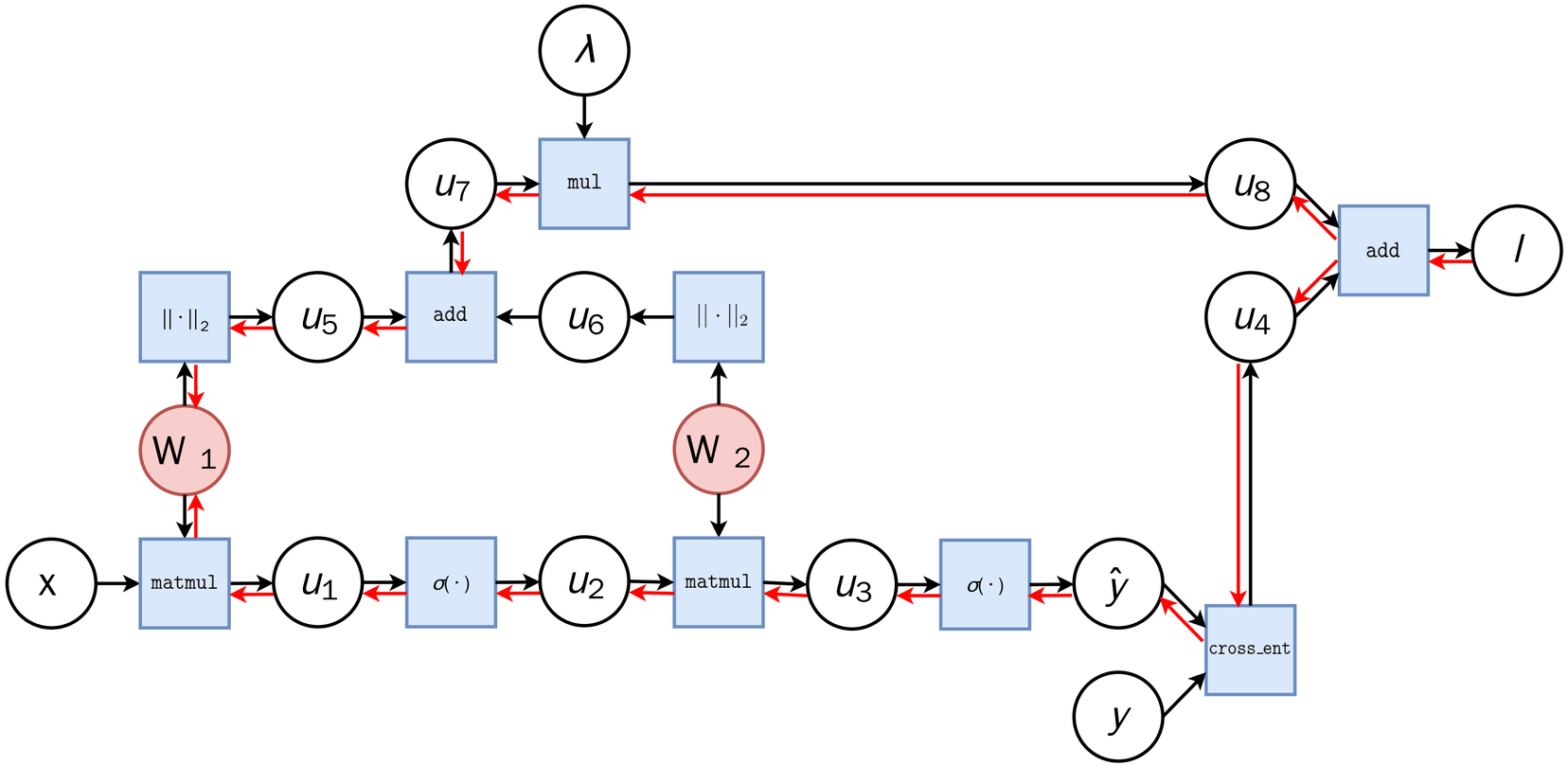
In the forward pass, intermediate values are all computed from inputs to outputs, which results in the annotated computational graph below:

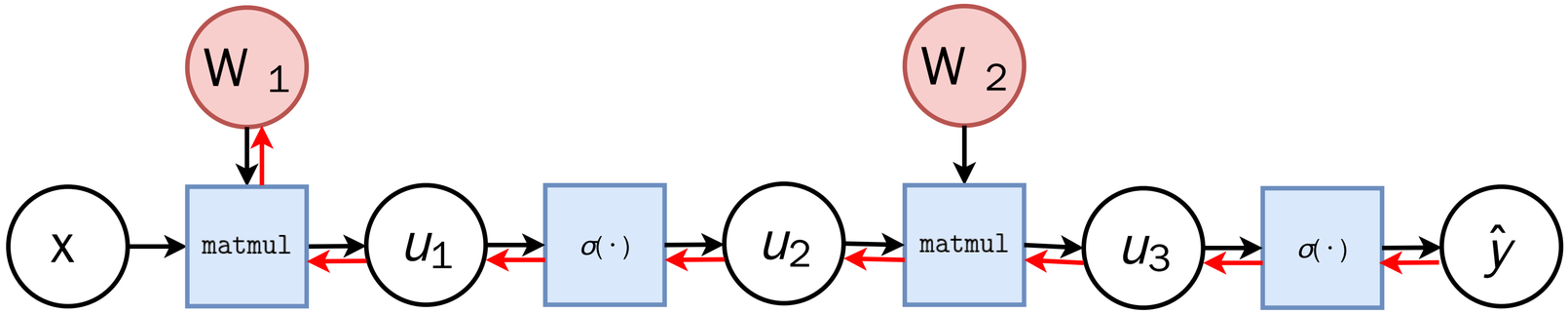


The total derivative can be computed through a backward pass, by walking through all paths from outputs to parameters in the computational graph and accumulating the terms. For example, for $\frac{\mathrm{d}\ell}{\mathrm{d}\mathbf{W}_1}$ we have:

$$\frac{\mathrm{d}\ell}{\mathrm{d}\mathbf{W}_1} = \frac{\partial \ell}{\partial u_8}\frac{\mathrm{d}u_8}{\mathrm{d}\mathbf{W}_1} + \frac{\partial \ell}{\partial u_4}\frac{\mathrm{d}u_4}{\mathrm{d}\mathbf{W}_1}$$

$$\frac{\mathrm{d}u_8}{\mathrm{d}\mathbf{W}_1} = \dots$$





Let us zoom in on the computation of the network output $\hat{y}$ and of its derivative with respect to $\mathbf{W}_1$.

- Forward pass: values u_1, u_2, u_3 and $\hat{y}$ are computed by traversing the graph from inputs to outputs given $\mathbf{x}$, $\mathbf{W}_1$ and $\mathbf{W}_2$.
- Backward pass: by the chain rule we have

$$\begin{aligned}\frac{\mathrm{d}\hat{y}}{\mathrm{d}\mathbf{W}_1} &= \frac{\partial \hat{y}}{\partial u_3}\frac{\partial u_3}{\partial u_2}\frac{\partial u_2}{\partial u_1}\frac{\partial u_1}{\partial \mathbf{W}_1} \\ &= \frac{\partial \sigma(u_3)}{\partial u_3}\frac{\partial \mathbf{W}_2^T u_2}{\partial u_2}\frac{\partial \sigma(u_1)}{\partial u_1}\frac{\partial \mathbf{W}_1^T u_1}{\partial \mathbf{W}_1}\end{aligned}$$

  Note how evaluating the partial derivatives requires the intermediate values computed forward.

This generalization allows to compose and design complex networks of operators, possibly dynamically, dealing with images, sound, text, sequences, etc. and to train them end-to-end.

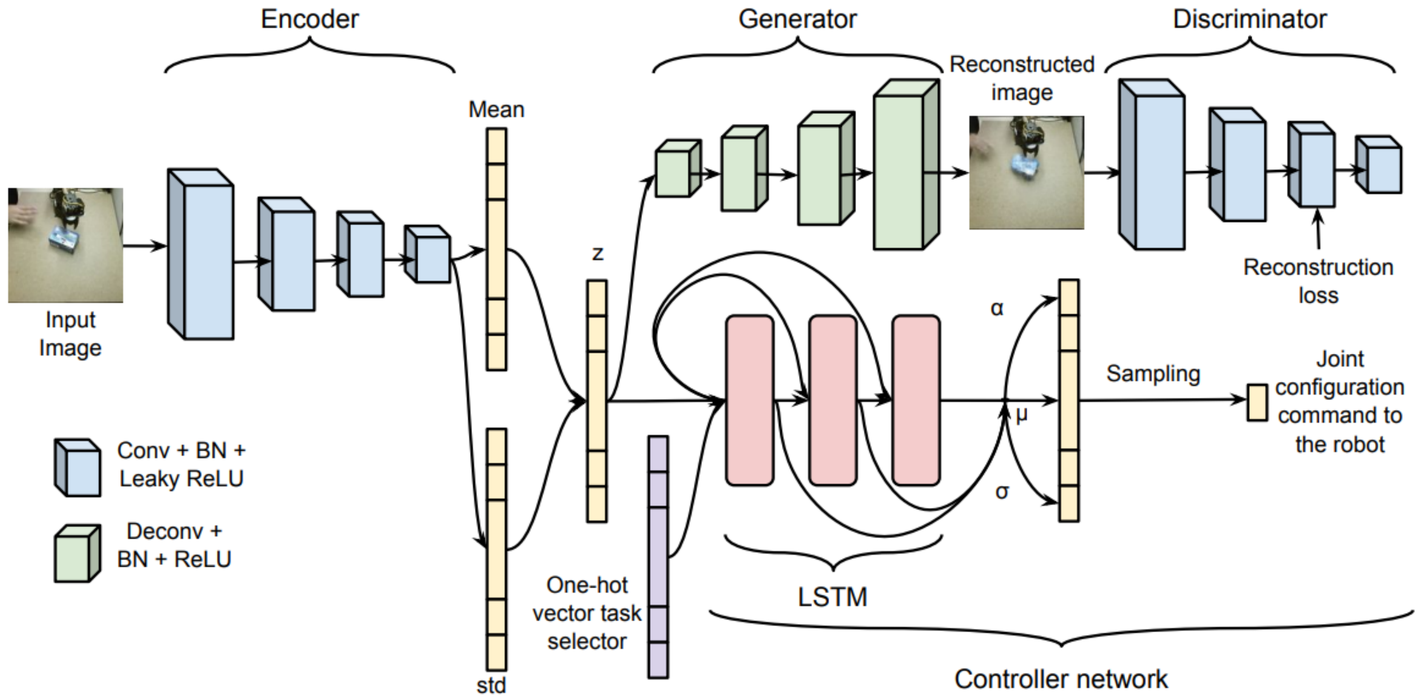
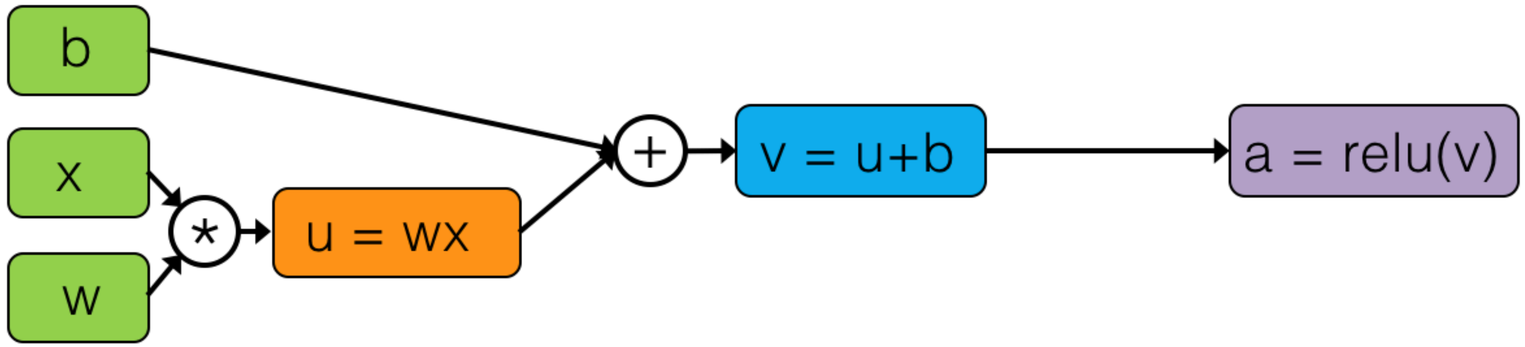
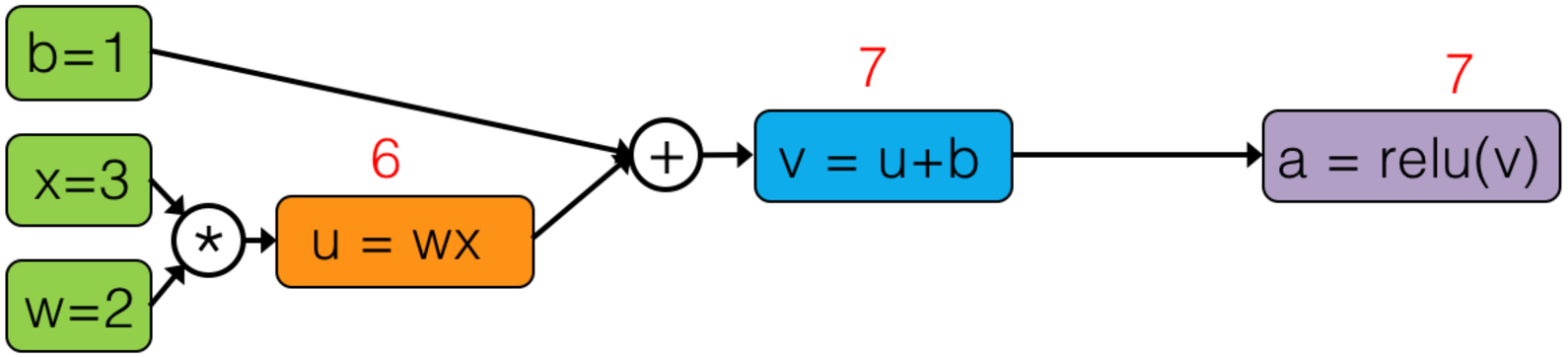


Fig. 2: Our proposed architecture for multi-task robot manipulation learning. The neural network consists of a controller network that outputs joint commands based on a multi-modal autoregressive estimator and a VAE-GAN autoencoder that reconstructs the input image. The encoder is shared between the VAE-GAN autoencoder and the controller network and extracts some shared features that will be used for two tasks (reconstruction and controlling the robot).

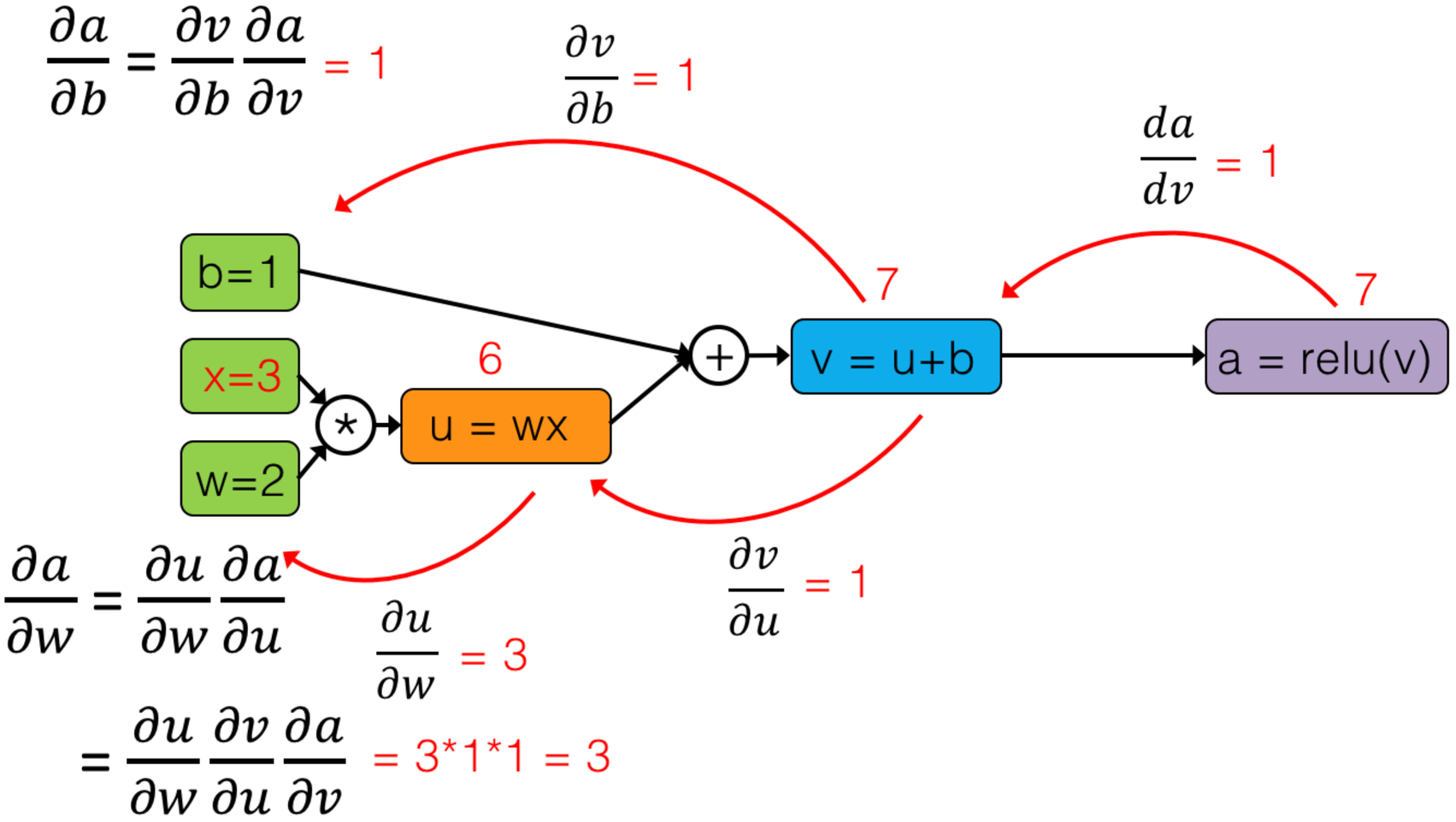
Credits: Francois Fleuret, EE559 Deep Learning, EPFL; Rahmatizadeh et al, 2017, arXiv:1707.02920.

Autograd example





Autograd example



Practice

Open a notebook or go to Colaboratory

https://colab.research.google.com