**2.4 Tab 2** Erstellen eines Tensors mit drei Elementen, Datentyp float und Gradientenfähigkeit (gradient capability).

Gradient capability = Tensor speichert und berechnet Gradienten, die für Backpropagation benötigt werden.

Backpropagation = Parameter werden angepasst, um Loss-Funktion zu minimieren.

By default ist requires\_grad in PyTorch false => wenn auf True gesetzt, werden Gradienten der Loss-Funktion automatisch berechnet

**2.4 Tab 3**

$$y = \frac{1}{|x|}\sum\_i \left[(x\_i + 2)^2 + 3\right]$$

Tab 4: Die backward()-Funktion wird benutzt um die Gradienten des Tensors y zu berechnen.

**2.5 Tab 1**; CUDA (Compute Unified Device Architecture) ist Schnittstelle von NVIDIA mit der Programmteile durch GPU abgearbeitet werden können, was Parallel Computing ermöglicht.

**2.6 Tab. 1** Quelle

**2.7 Tab. 2**

|  |  |  |
| --- | --- | --- |
| X\_1 | X\_2 | Output |
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

**3.1** Loss Models, loss function

Loss-Funktion berechnet Differenz zwischen Vorhersagen des Netzwerks und „echten“ Labels. In PyTorch sehr einfach mit vordefinierten Loss-Funktionen möglich.

Hier binäres Klassifikationsproblem, daher Benutzung von Binary Cross Entropy (BCE)

**3.2** Gradientenverfahren

Parameter sollen so angepasst werden, dass der Loss minimiert wird.

In PyTorch sind im torch.optim-Package viele Optimizer vordefiniert.

Einfachster Optimizer ist Stochastic Gradient Descent (SGD).

Learning Rate (lr) wird auf 0.1 gesetzt.