Universität Stuttgart
Institut für Maschinelle Sprachverarbeitung

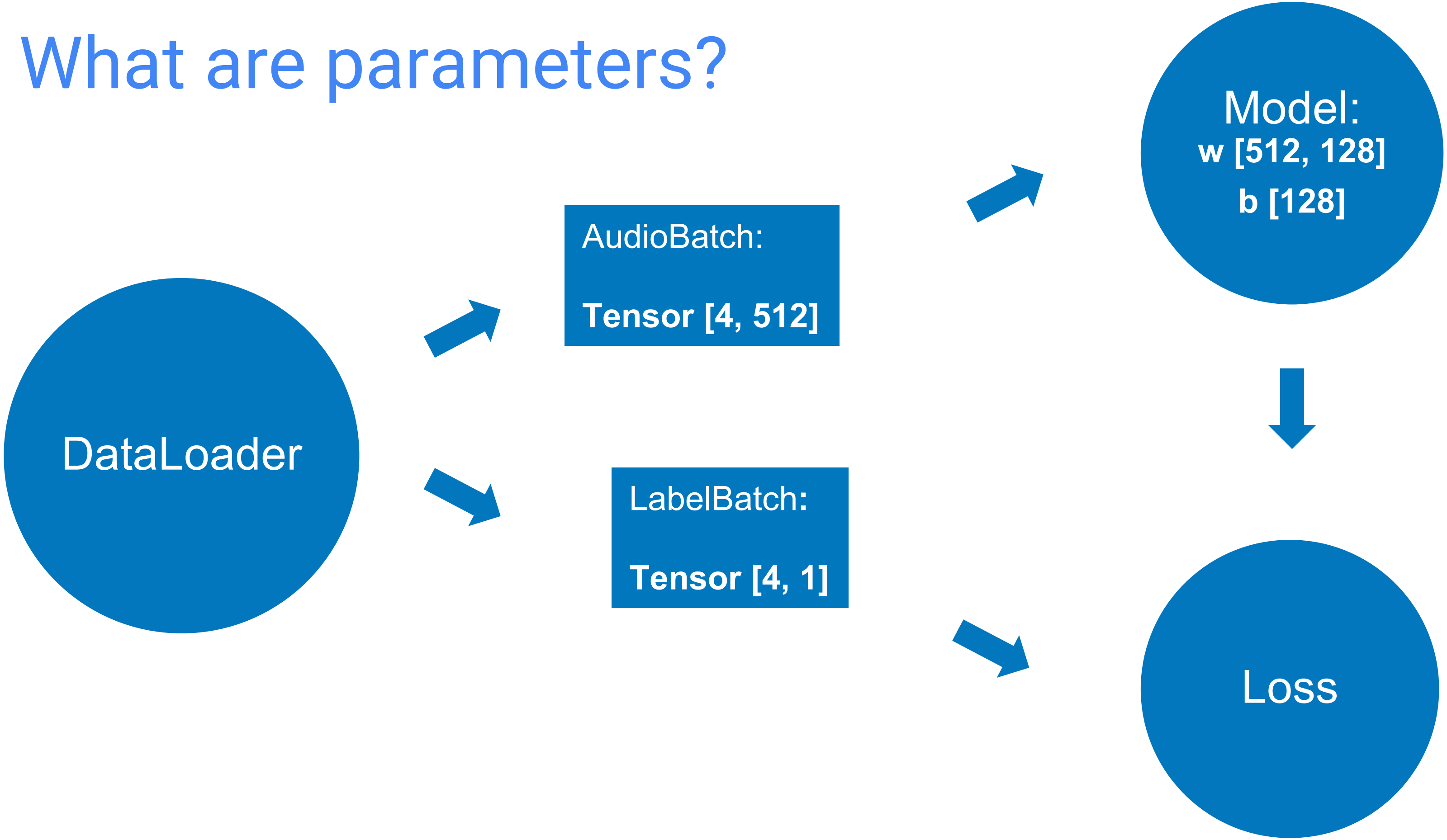
Yixuan Xiao

Neural Nets

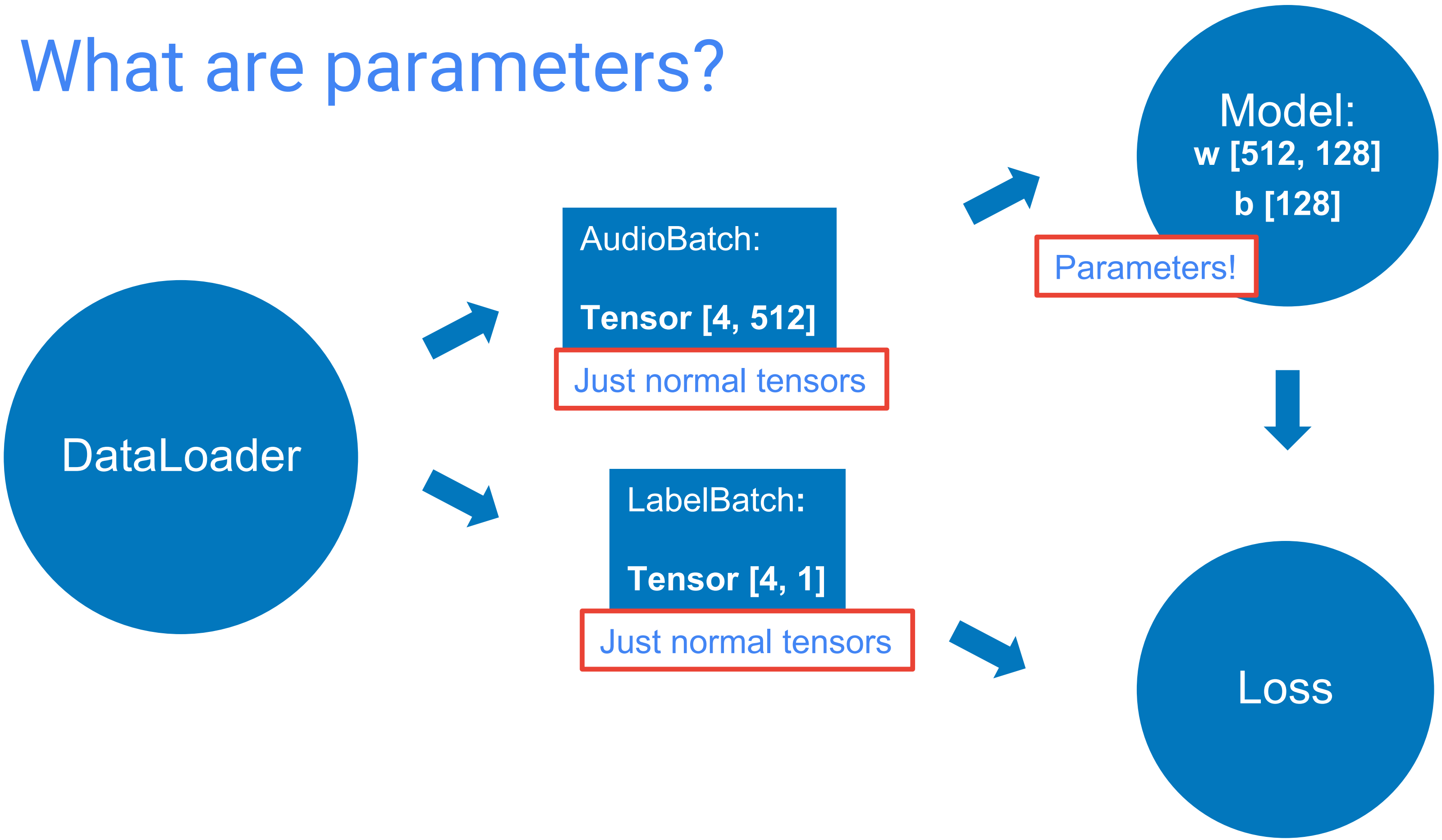
PyTorch – Neural Nets

- How to use `nn.Linear` to build a simple multi-layer neural net?
  - Parameter tensors and normal tensors
  - `torch.nn.Module`
  - `nn.Linear`
- What is `Loss` and how to train you model using `loss.backward()`
- (Advanced-Optional/Time Permit) : Computation Graph
  - Won’t be in the exam
  - If you’re interested in the low-level implementation, can take a look

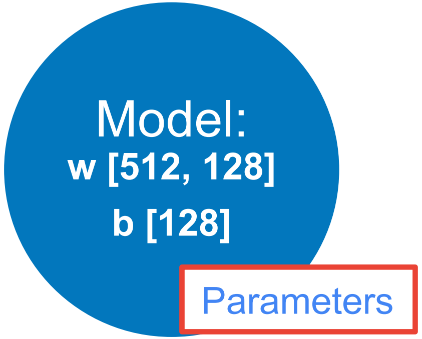
What are parameters?



What are parameters?

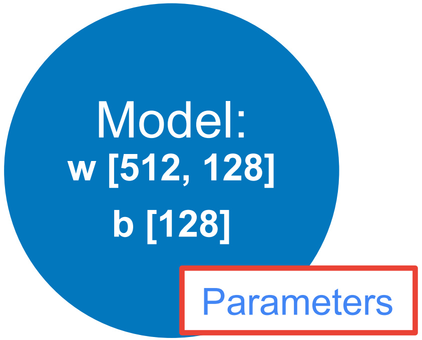


torch.nn.module



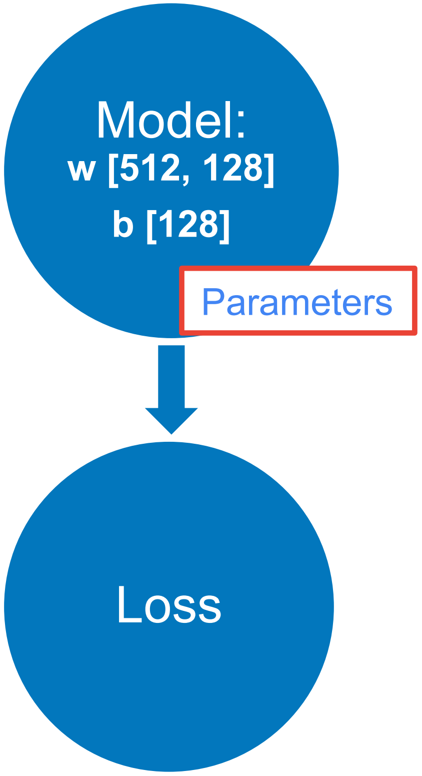
- All models belongs to torch.nn.Module
- Recap: All datasets belong to torch.util.data.dataset
- So to create your own model, should import Module and code: `MyModel(Module)

torch.nn.module



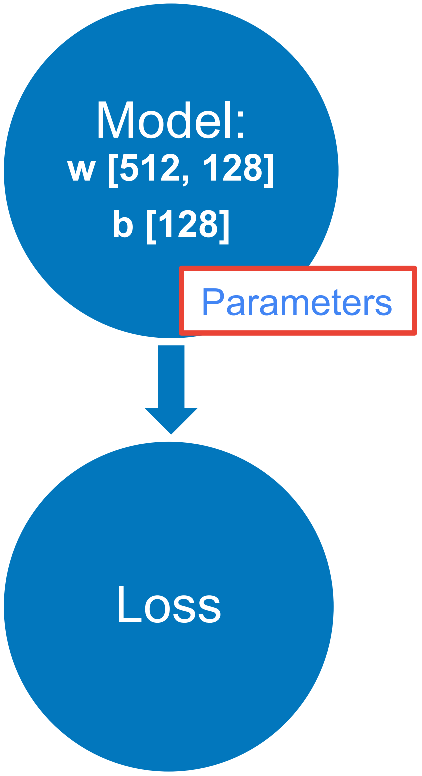
- `__init__`: define what your model is, e.g., a CNN + a linear project, a LSTN+CNN+Projection
- `forward()`: define the forward pass
- Other useful functions:
  - `named_parameters`: show all parameter tensors inside
  - `load_state_dict`: load a saved model
  - `train()` and `eval()`: whether you're using that in a training model or inference mode

torch.nn.module



- If a component needs `forward` and `backward`, then it needs to be a `torch.nn.module`
- Question: Is Loss a module?

torch.nn.module



- If a component needs `forward` and `backward`, then it needs to be a `torch.nn.module`
- Question: Is Loss a module?

- Yes! Loss is a module.
- Recap: backpropagation, backward gradient from loss to the input.

Let’s Code

Links

- `nn.Linear`: https://docs.pytorch.org/docs/stable/generated/torch.nn.Linear.html
- `activation functions`: https://docs.pytorch.org/docs/stable/nn.html#non-linear-activations-weighted-sum-nonlinearity
- `loss`: https://docs.pytorch.org/docs/stable/nn.html#loss-functions

Computation graph

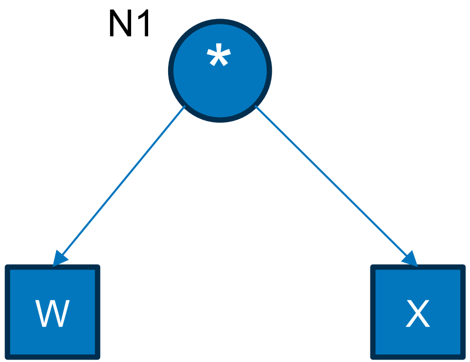
- Y = sigmoid(WX+b)
- Computation order

- N1: W * X
- N2: N1 + b
- N3: sigmoid(N3)

Computation graph

- Y = sigmoid(WX+b)
- Computation order

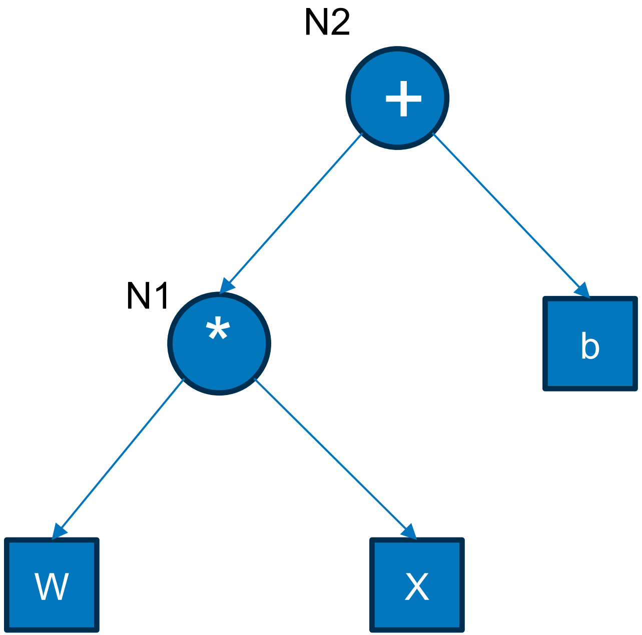
- N1: W * X



Computation graph

- Y = sigmoid(WX+b)
- Computation order

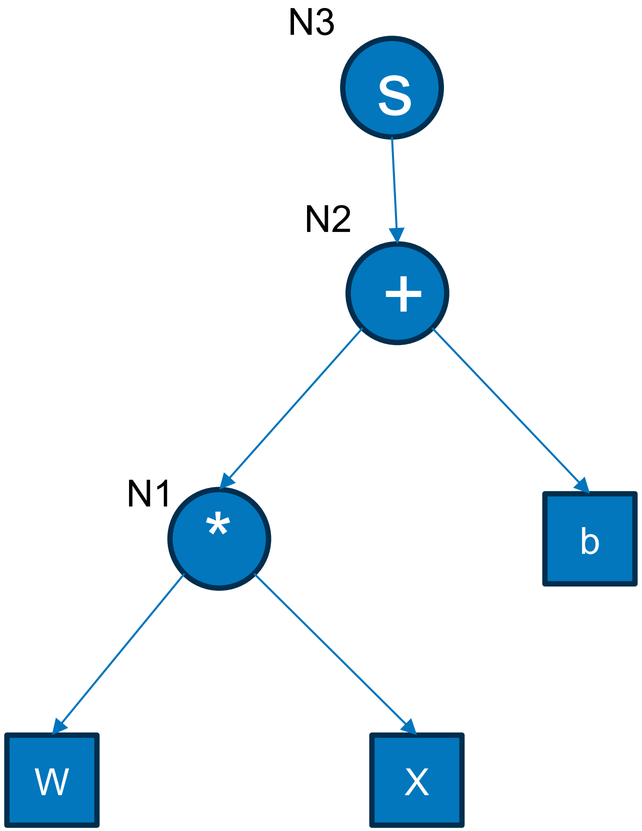
- N1: W * X
- N2: N1 + b



Computation graph

- Y = sigmoid(WX+b)
- Computation order

- N1: W * X
- N2: N1 + b
- N3: sigmoid(N2)

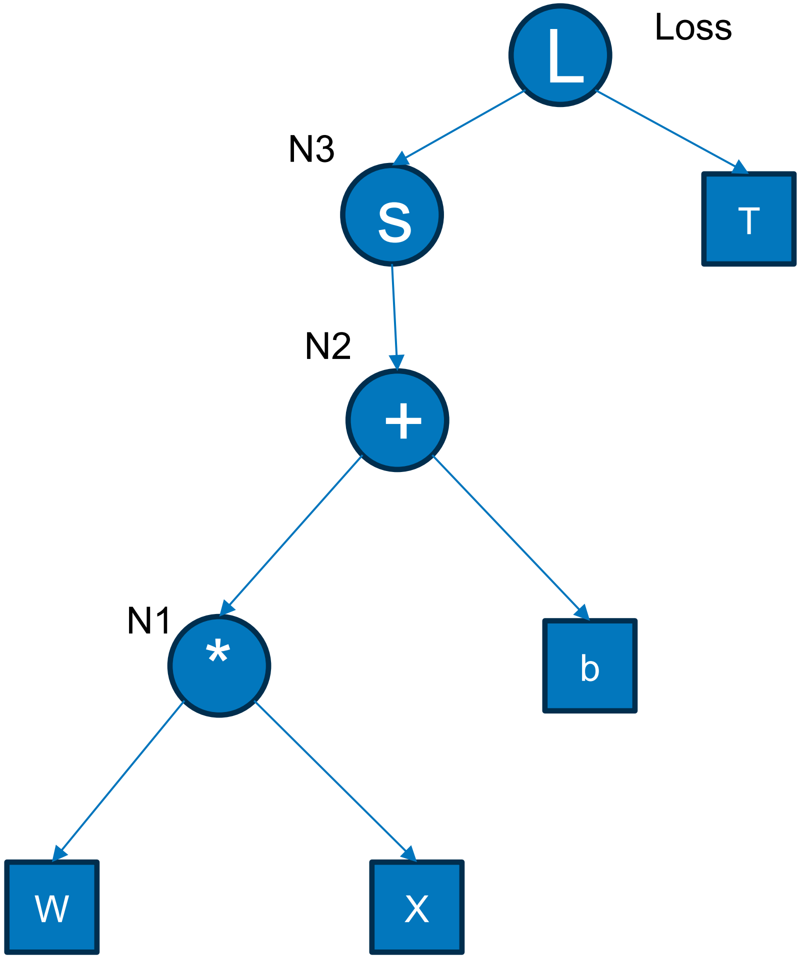


Computation graph

- Y = sigmoid(WX+b)
- Computation order

- N1= W * X
- N2 = N1 + b
- N3 = sigmoid(N2)

- Loss (N3, target)



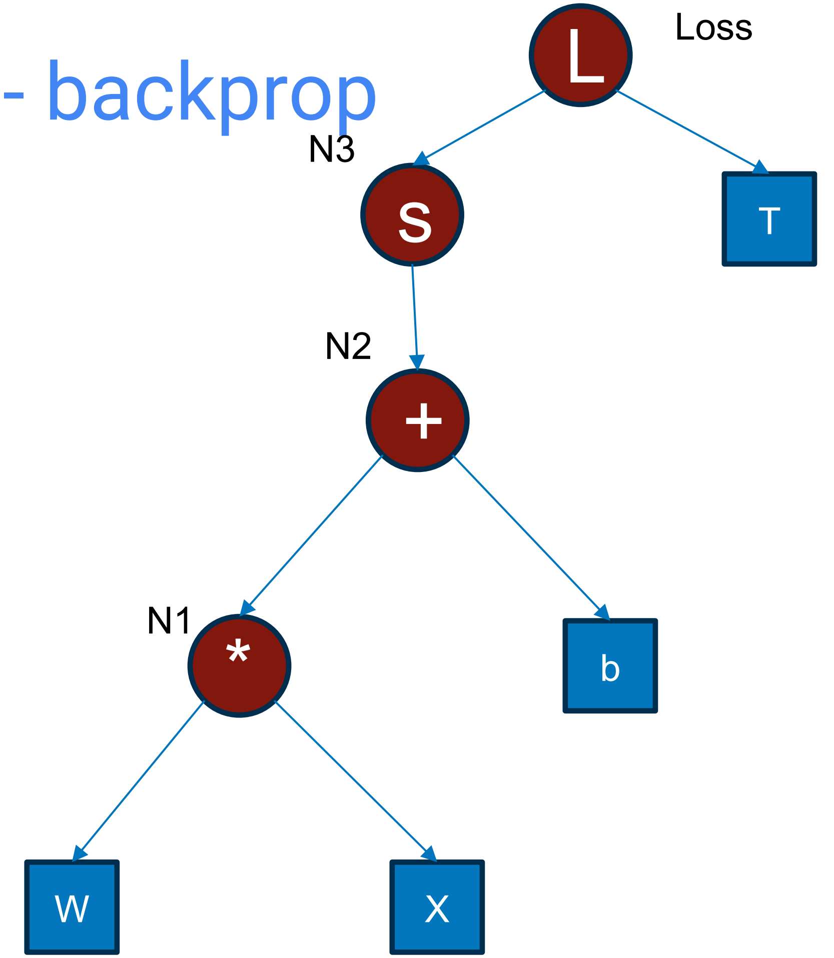
Computation graph - backprop

- Y = sigmoid(WX+b)
- Computation order

- N1: W * X
- N2: N1 + b
- N3: sigmoid(N2)

- Loss (N3, target)

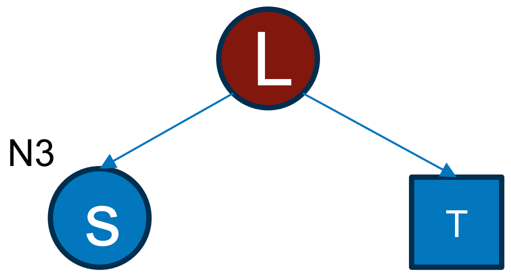
- Reverse the order
- Only do that for **operator**



Computation graph - backprop

- **Loss** (S, T)

- Compute local gradient:
- $\frac{\partial L}{\partial \mathrm{N3}}, \frac{\partial L}{\partial T}$



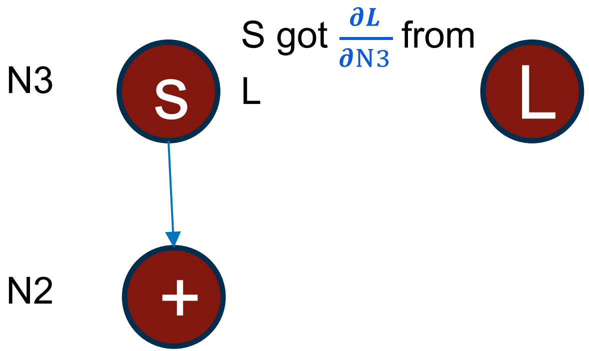
- Because L is the final output, we don't need to apply chain rules
- Final outputs: $\frac{\partial L}{\partial \mathrm{N3}}$ **(given to S),** $\frac{\partial L}{\partial T}$(given to T, but T is not an operator)

Computation graph - backprop

- Operator: **S**(+)
- N3 = sigmoid(N2)

- Compute local gradient:
- $\frac{\partial N3}{\partial N2}$

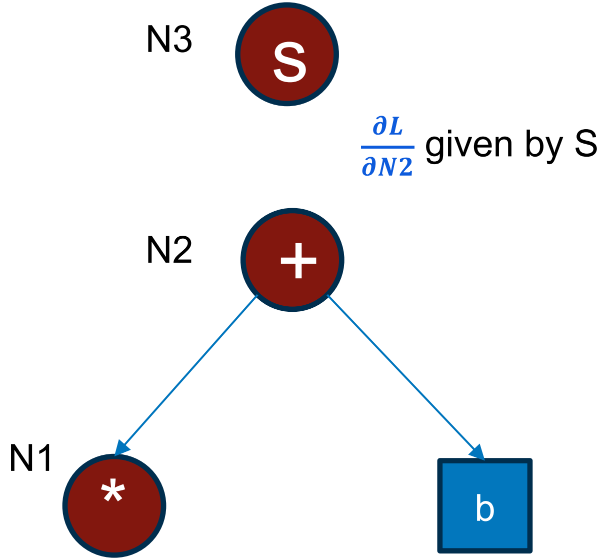
- Chain rules: $\frac{\partial L}{\partial N2} = \frac{\partial L}{\partial N3} * \frac{\partial L}{\partial N2}$
- **Operator S gives $\boldsymbol{\frac{\partial L}{\partial N2}}$ to operator +**



Computation graph - backprop

- Operator: **+**
- N2 = N1 + b

- Compute local gradient:
- $\frac{\partial \mathrm{N2}}{\partial \mathrm{N1}}, \frac{\partial \mathrm{N2}}{\partial \mathrm{b}}$

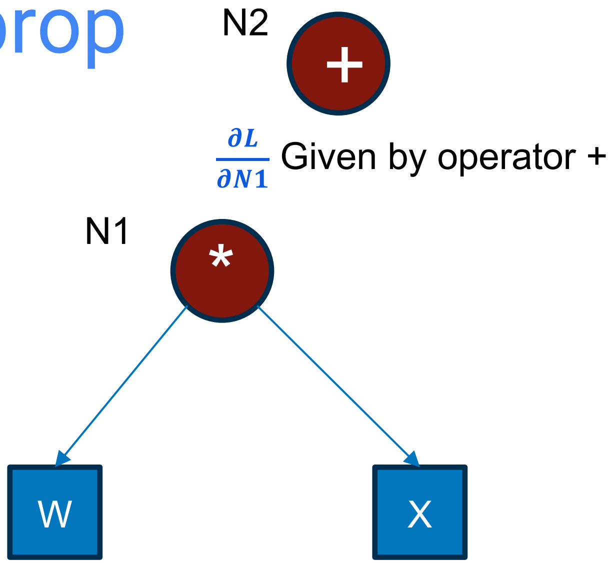


- Chain rules: $\frac{\partial \mathrm{L}}{\partial \mathrm{b}} = \frac{\partial \mathrm{L}}{\partial \mathrm{N2}} * \frac{\partial \mathrm{N2}}{\partial \mathrm{b}}$ can update b now!
- Chain rules: $\boldsymbol{\frac{\partial L}{\partial N1}} = \frac{\partial \mathrm{L}}{\partial \mathrm{N2}} * \frac{\partial N2}{\partial N1}$ **Given to operator ***

Computation graph - backprop

- Operator: *
- N1 = W*X

- Compute local gradient:
- $\frac{\partial \mathrm{N1}}{\partial \mathrm{W}}, \frac{\partial \mathrm{N1}}{\partial \mathrm{X}}$



- Chain rules: $\frac{\partial L}{\partial \mathrm{W}} = \frac{\partial \mathrm{L}}{\partial N1} * \frac{\partial \mathrm{N1}}{\partial \mathrm{W}}$ Can update W now!