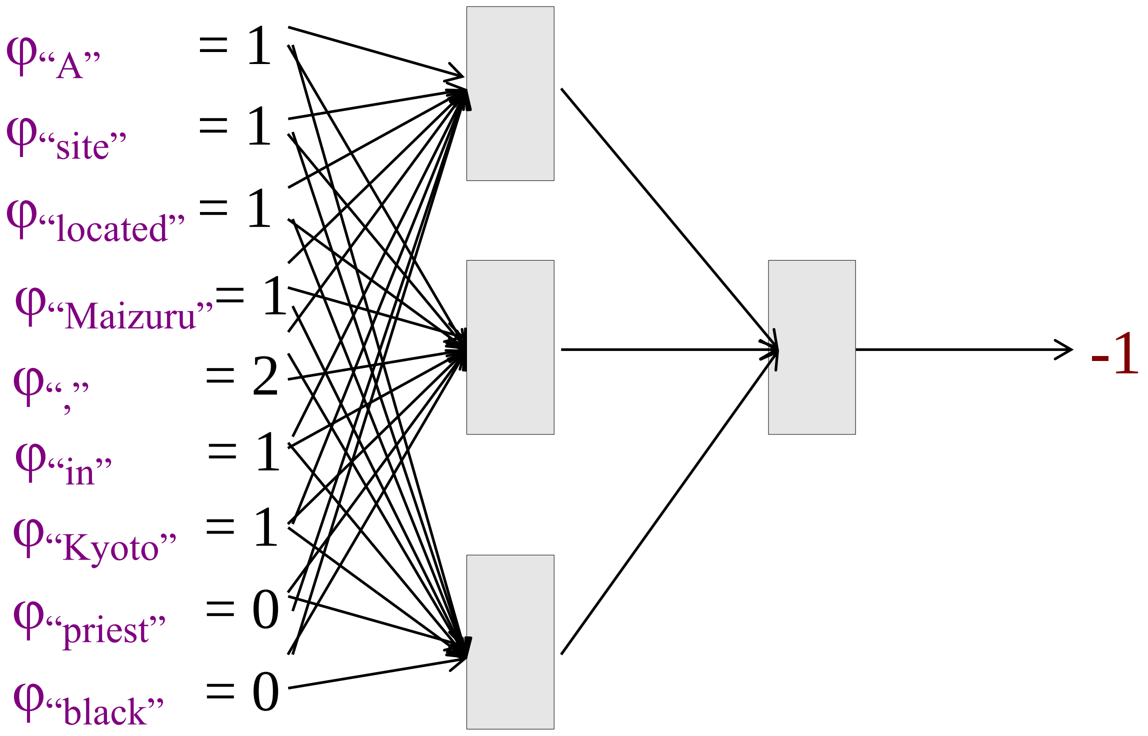
COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

Neural Networks, Computation Graphs

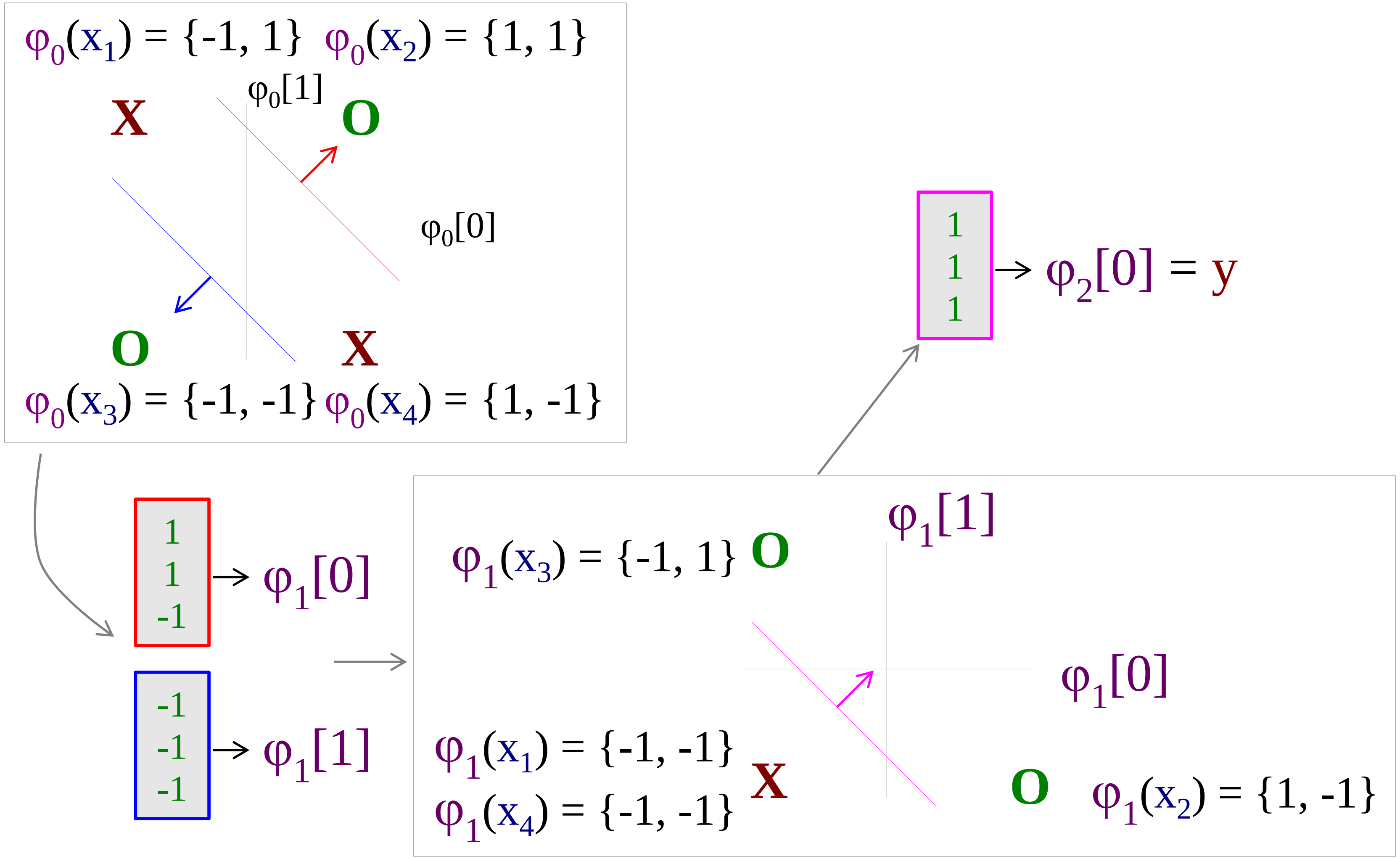
CMSC 470

Marine Carpuat

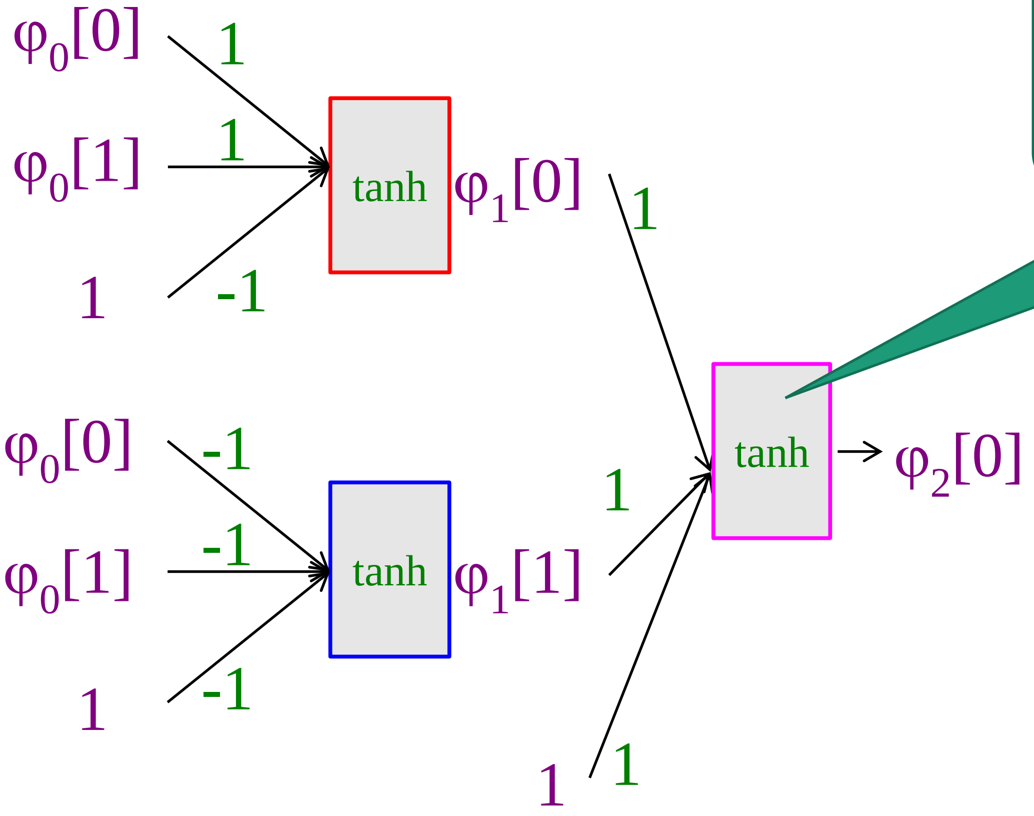
Binary Classification with a Multi-layer Perceptron



Example: binary classification with a NN

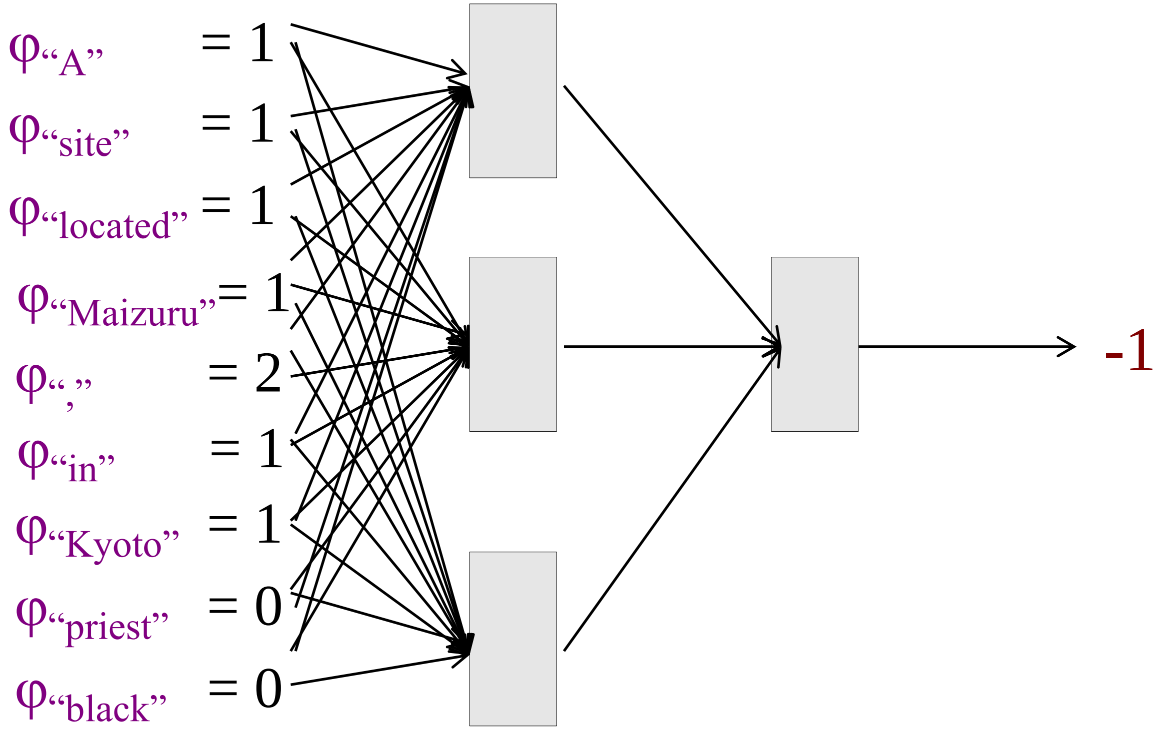


Example: the Final Net



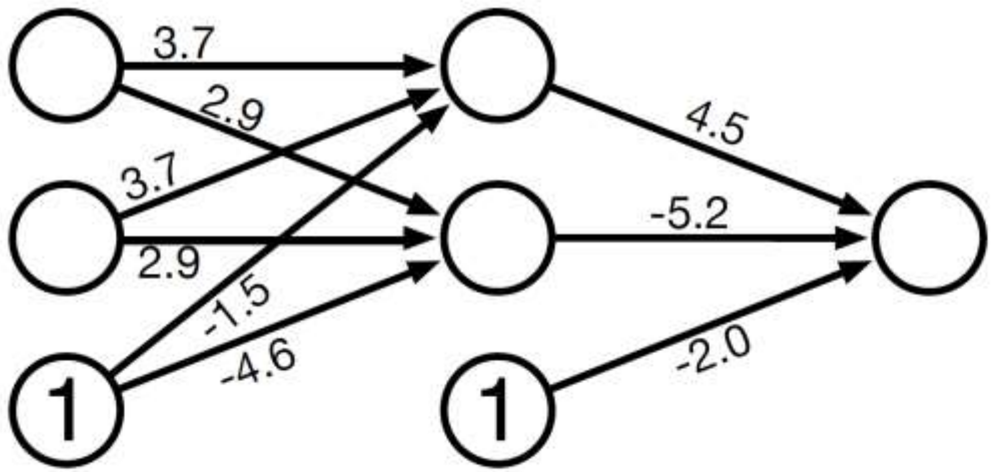
Replace “sign” with smoother non-linear function (e.g. tanh, sigmoid)

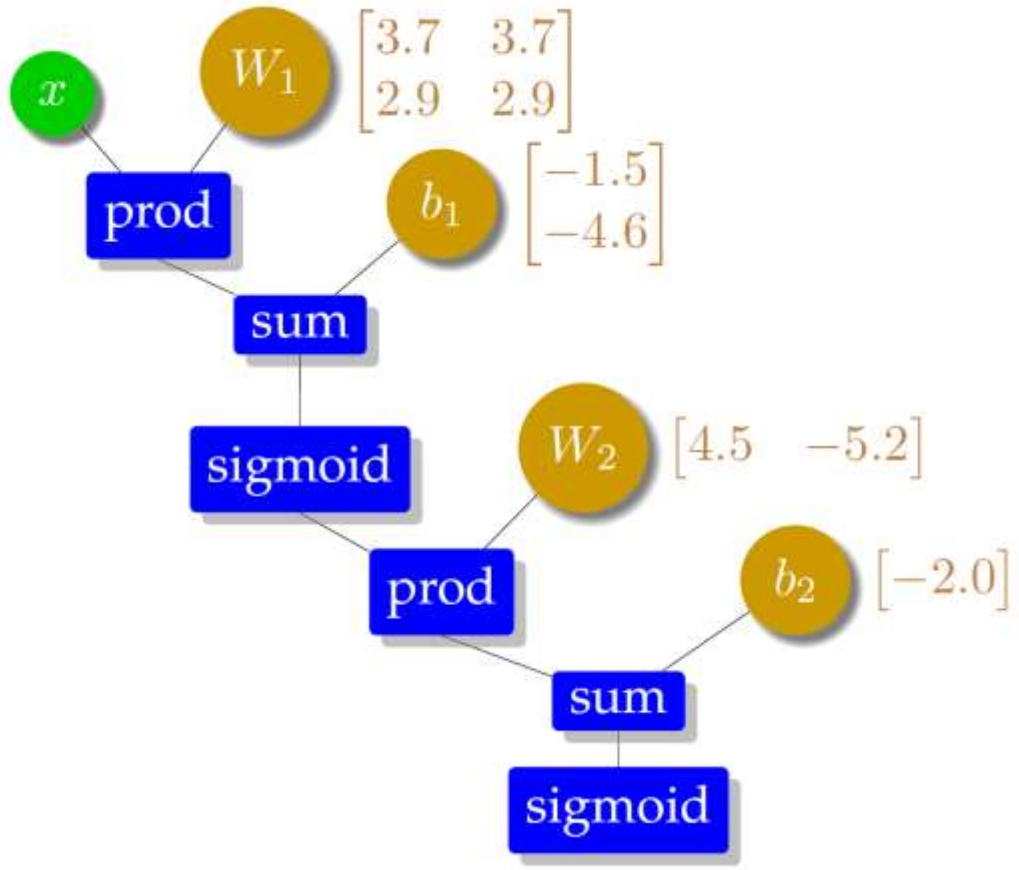
Multi-layer Perceptrons are a kind of “Neural Network” (NN)



- Input (aka features)
- Output
- Nodes (aka neuron)
- Layers
- Hidden layers
- Activation function **(non-linear)**

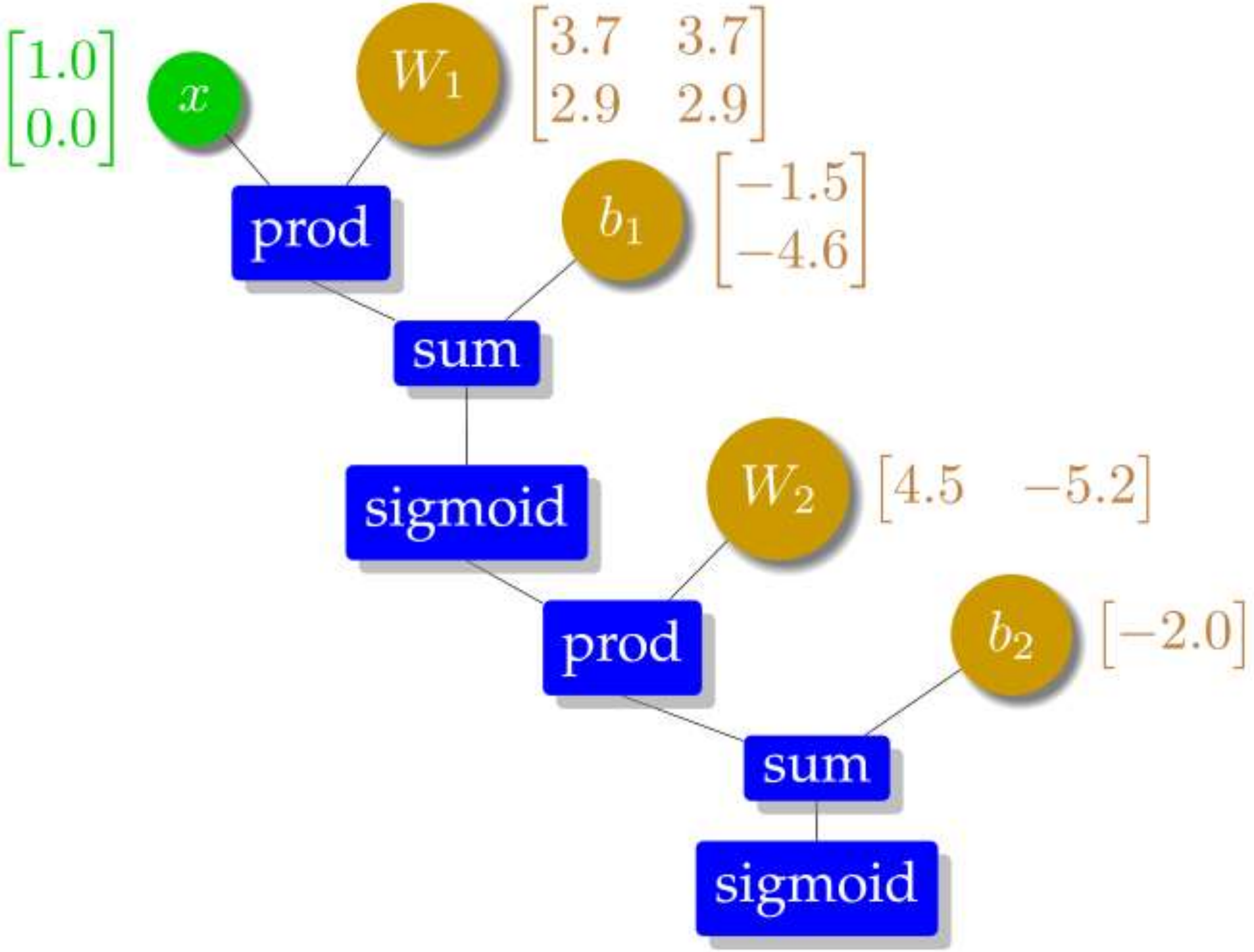
Neural Networks as Computation Graphs



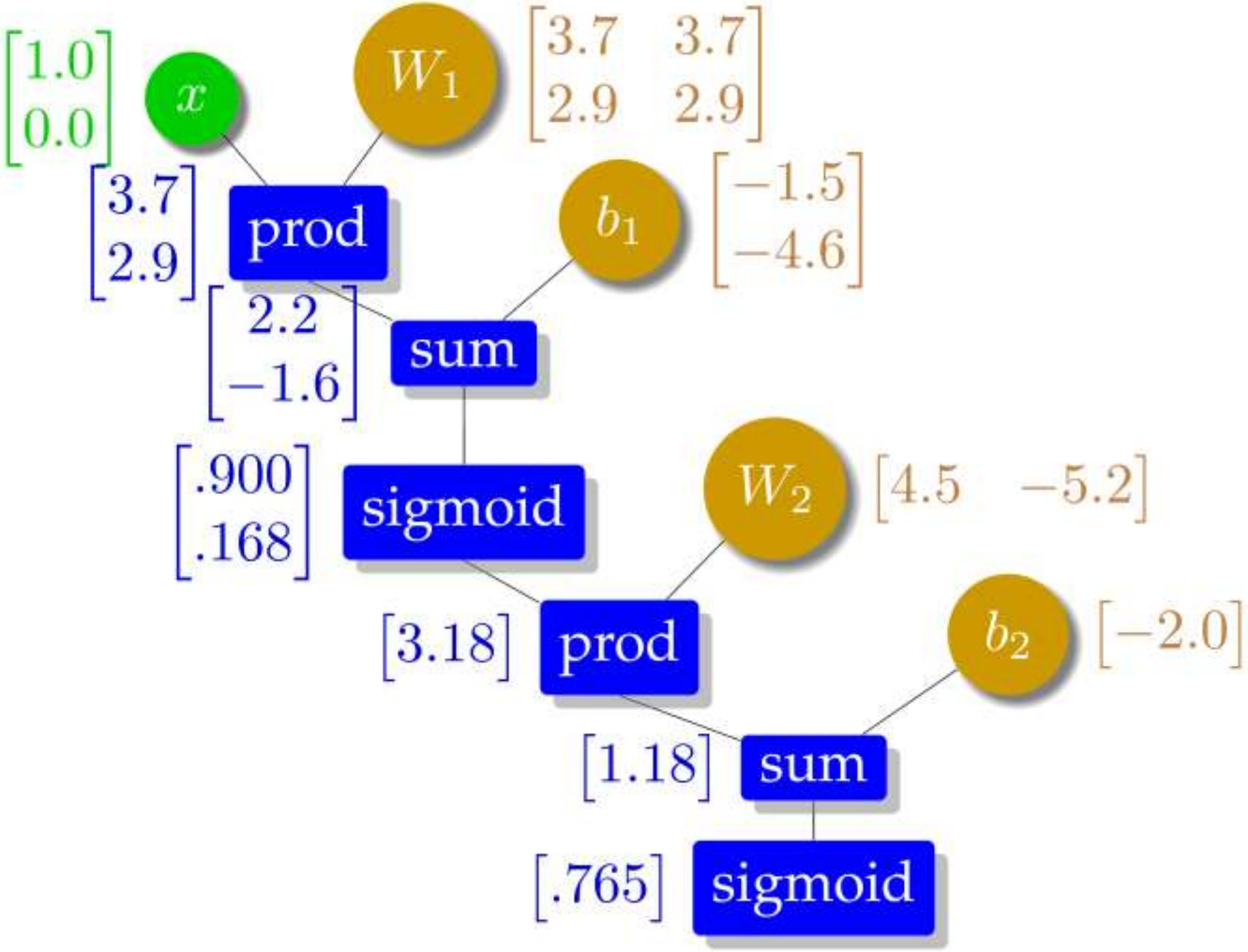


Example & figures by Philipp Koehn

Computation Graphs Make Prediction Easy: Forward Propagation



Computation Graphs Make Prediction Easy: Forward Propagation



Neural Networks as Computation Graphs

- Decomposes computation into simple operations over matrices and vectors

- Forward propagation algorithm
  - Produces network output given an output
  - By traversing the computation graph in topological order

Neural Networks for Multiclass Classification

Multiclass Classification

- The softmax function

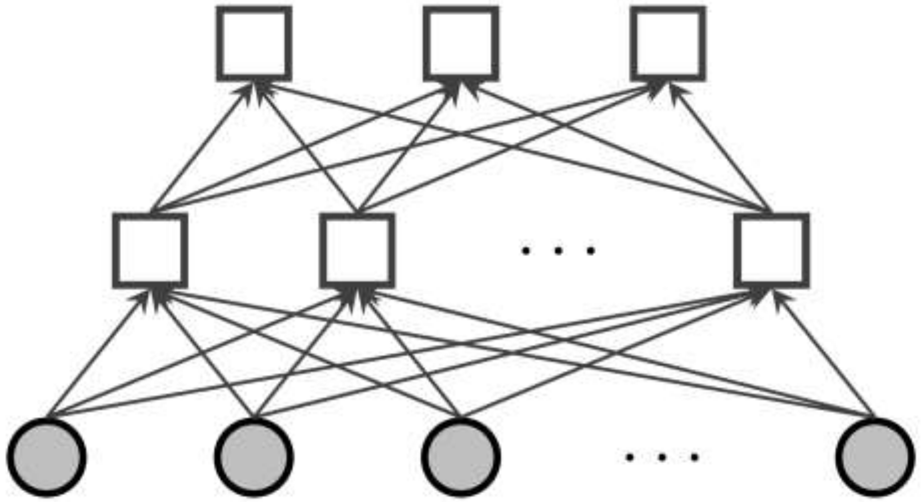
$$P(y \mid x) = \frac{e^{\mathbf{w}\cdot\phi(x,y)}}{\sum_{\tilde{y}} e^{\mathbf{w}\cdot\phi(x,\tilde{y})}}$$

← Current class

← Sum of other classes

Exact same function as in multiclass logistic regression

Example: A feedforward Neural Network for 3-way Classification



Sigmoid function

$$\boldsymbol{z} = \sigma(\boldsymbol{\Theta}^{(x \to z)} \boldsymbol{x})$$

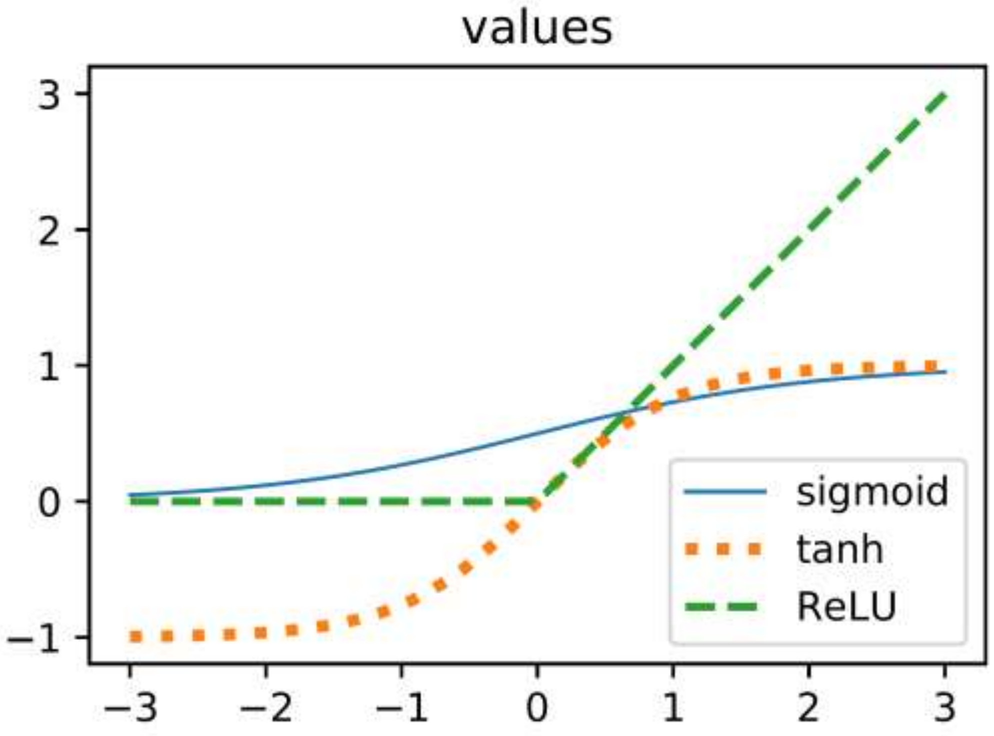
$$\mathrm{p}(y \mid \boldsymbol{x}; \boldsymbol{\Theta}^{(z \to y)}, \boldsymbol{b}) = \mathrm{SoftMax}(\boldsymbol{\Theta}^{(z \to y)} \boldsymbol{z} + \boldsymbol{b})$$

Softmax function (as in multi-class logistic reg)

From Eisenstein p66

Designing Neural Networks: Activation functions

- Hidden layer can be viewed as set of hidden features
- The output of the hidden layer indicates the extent to which each hidden feature is "activated" by a given input
- The activation function is a non-linear function that determines range of hidden feature values



Designing Neural Networks: Network structure

- 2 key decisions:
  - Width (number of nodes per layer)
  - Depth (number of hidden layers)

- More parameters means that the network can learn more complex functions of the input

Neural Networks so far

- Powerful non-linear models for classification
- Predictions are made as a sequence of simple operations
  - matrix-vector operations
  - non-linear activation functions
- Choices in network structure
  - Width and depth
  - Choice of activation function
- Feedforward networks (no loop)
- Next: how to train?

Training Neural Networks

How do we estimate the parameters (aka "train") a neural net?

For training, we need:

- Data: (a large number of) examples paired with their correct class (x,y)
- Loss/error function: quantify how bad our prediction y is compared to the truth t
  - Let's use squared error:

$$\text{error} = \frac{1}{2}(t - y)^2$$

Stochastic Gradient Descent

- We view the error as a function of the trainable parameters, on a given dataset
- We want to find parameters that minimize the error

Start with some initial parameter values

$w = \mathbf{0}$

for *I* iterations

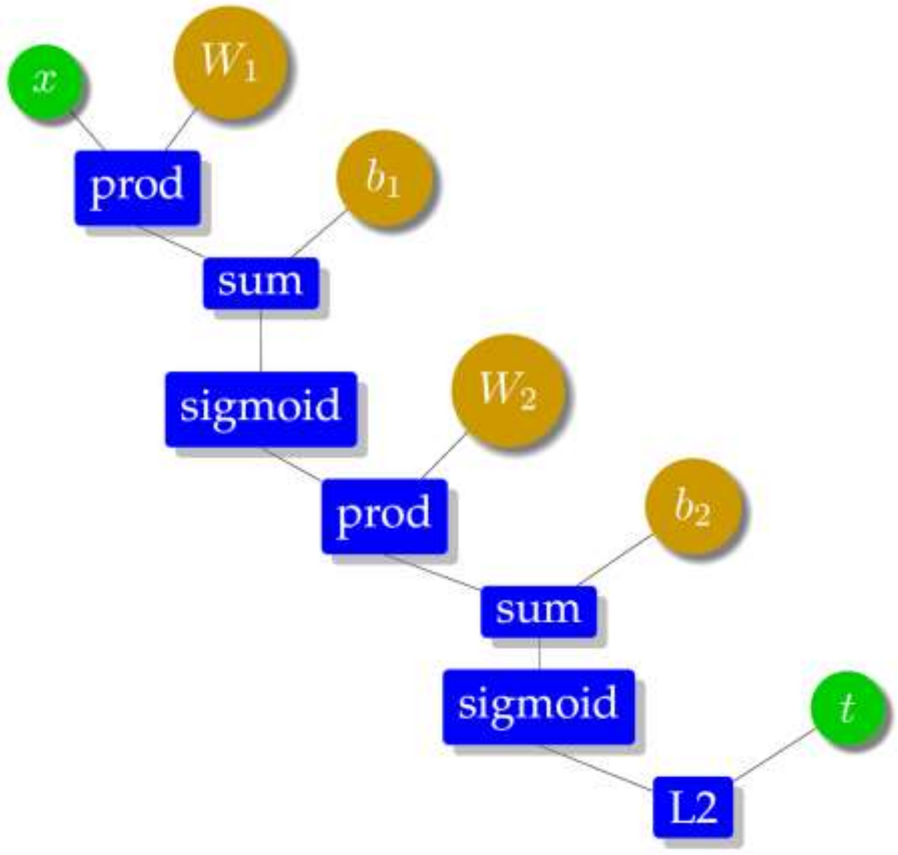
for each labeled pair *x*, *y* in the data

Go through the training data one example at a time

$$w = w - \mu \frac{d\,\mathrm{error}(w, x, y)}{dw}$$

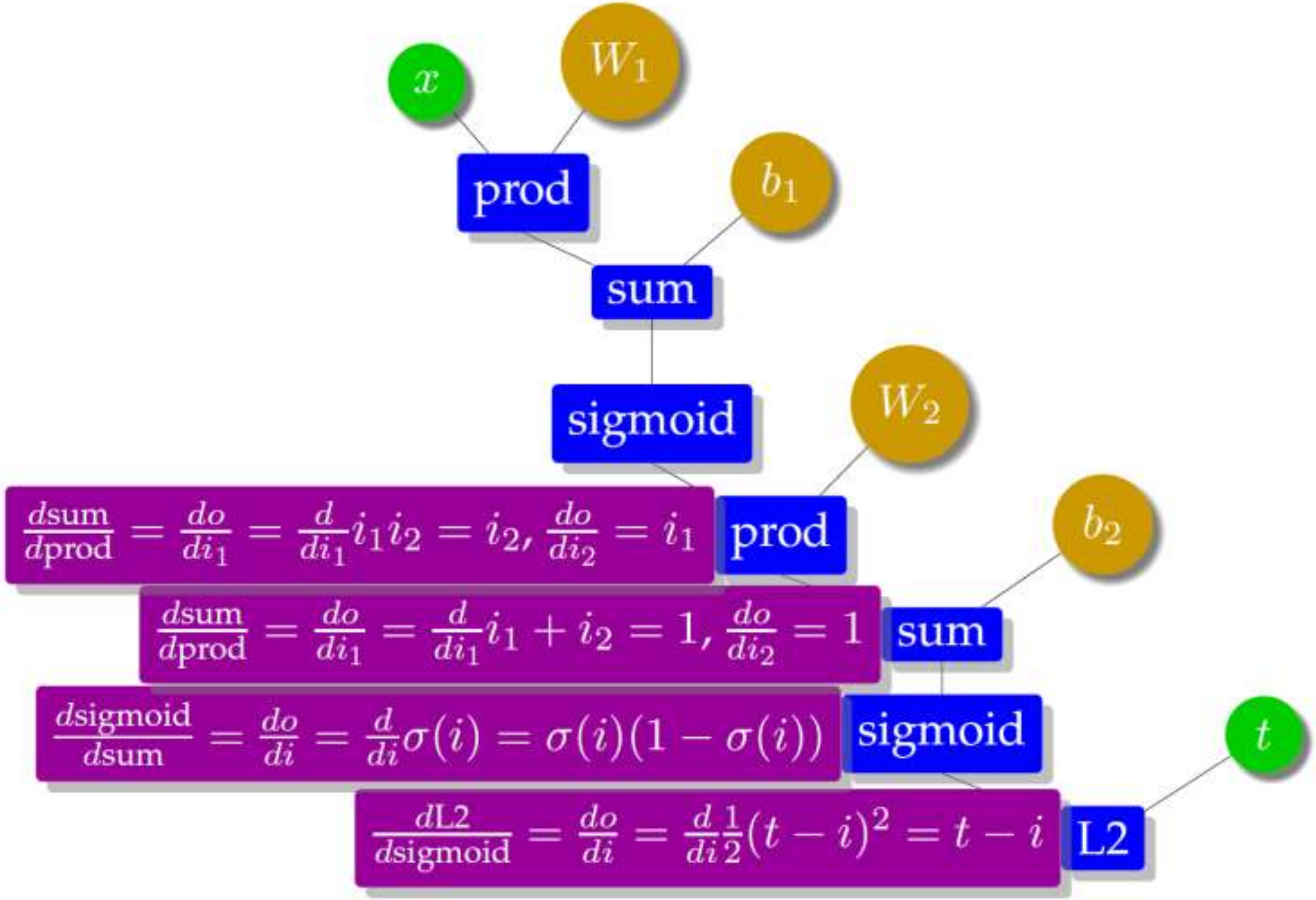
Take a step down the gradient

Computation Graphs Make Training Easy: Computing Error

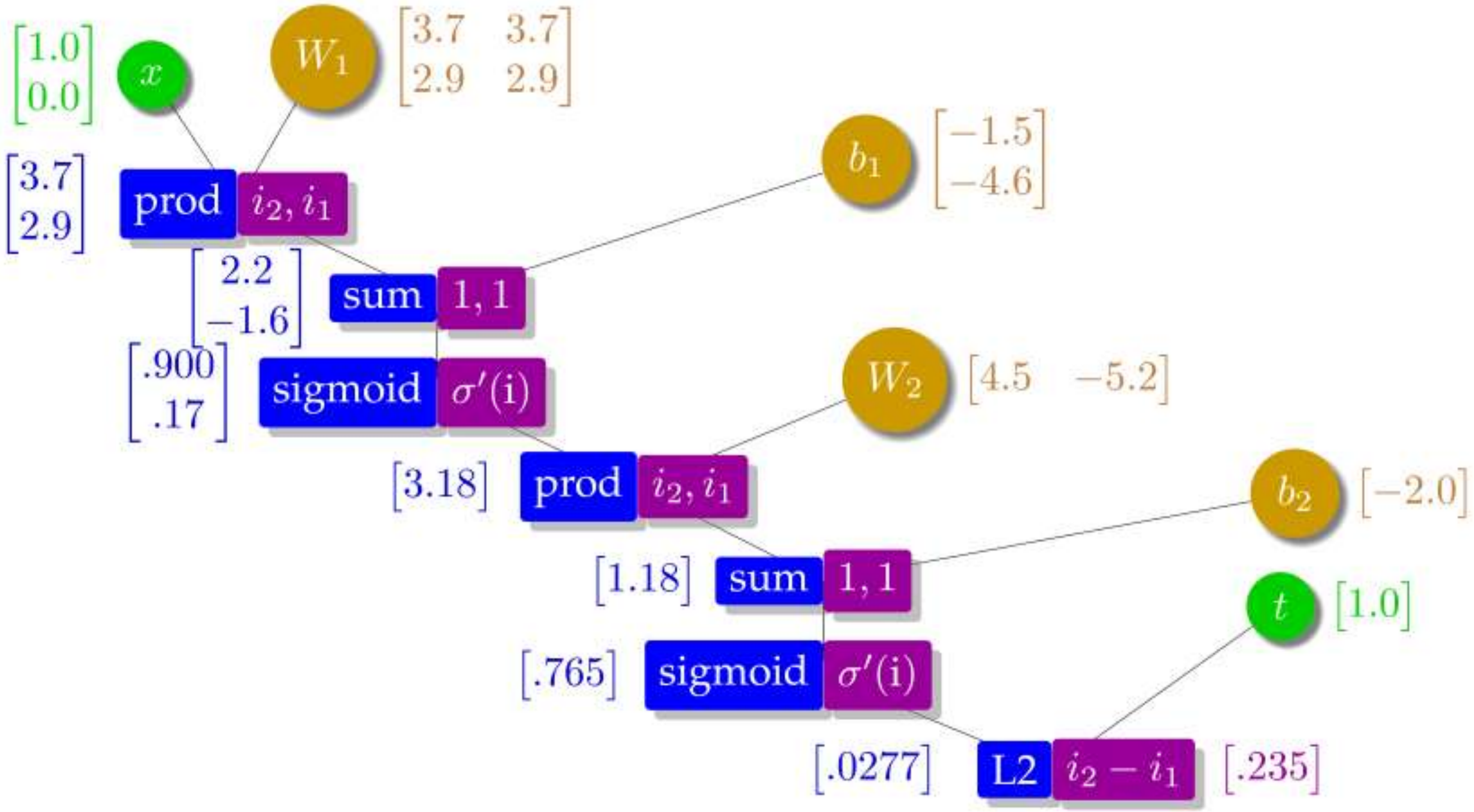


Computation Graphs Make Training Easy: Computing Gradients

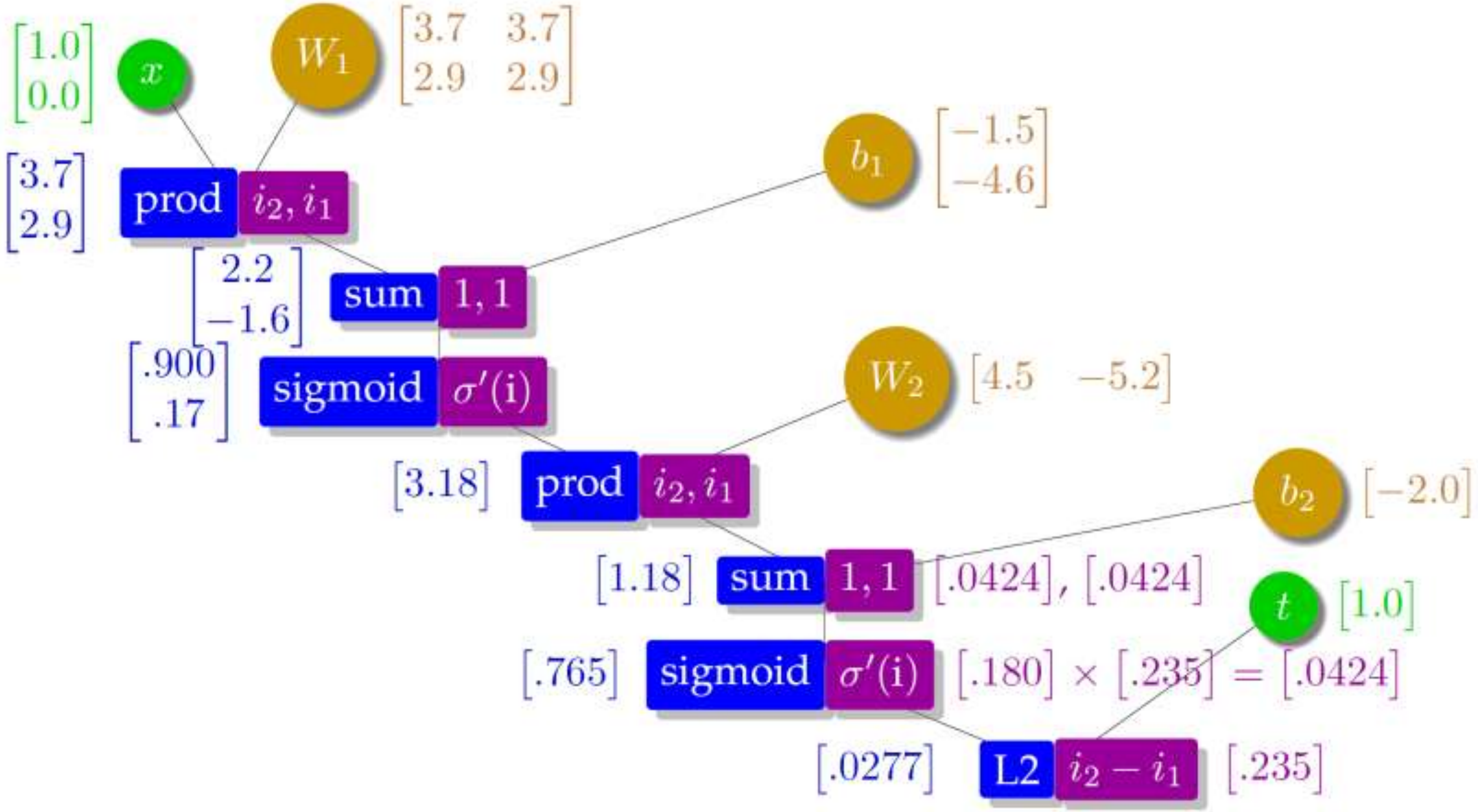
$$\frac{dE}{dA} = \frac{dE}{dB}\frac{dB}{dA}$$



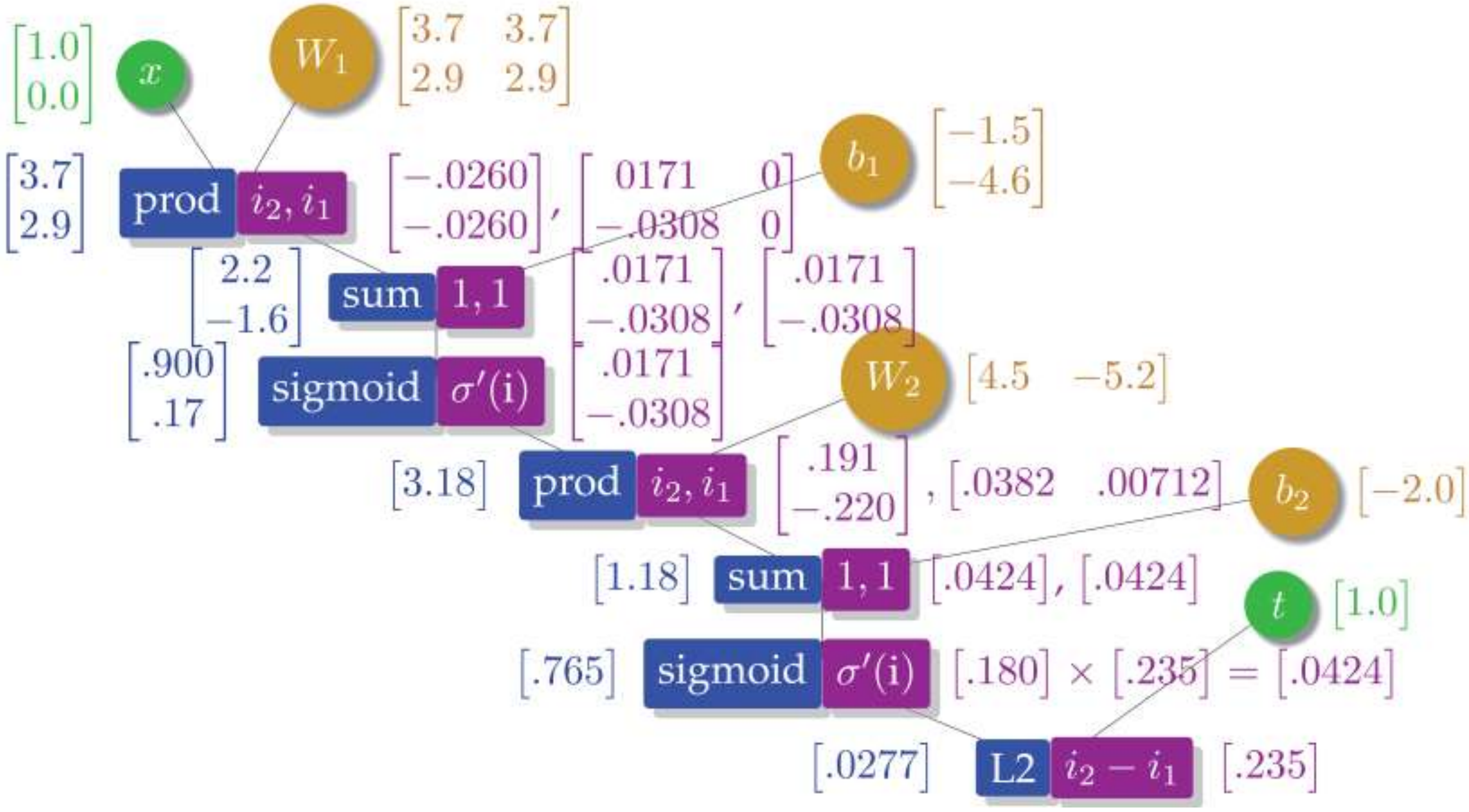
Computation Graphs Make Training Easy: Given forward pass + derivatives for each node



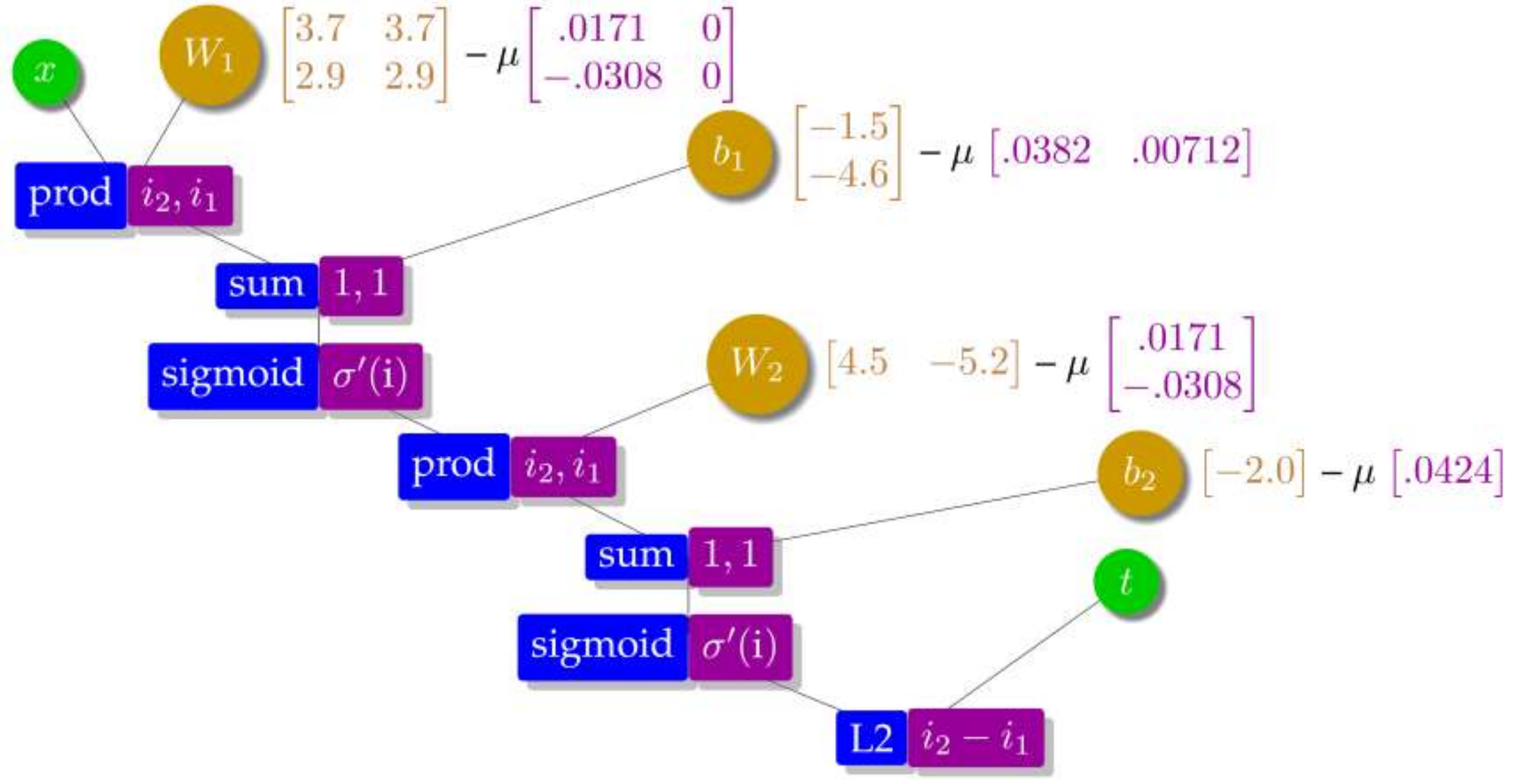
Computation Graphs Make Training Easy: Computing Gradients



Computation Graphs Make Training Easy: Computing Gradients



Computation Graphs Make Training Easy: Updating Parameters



Computation Graph: A Powerful Abstraction

- To build a system, we only need to:
  - Define network structure
  - Define loss
  - Provide data
  - (and set a few more hyperparameters to control training)

- Given network structure
  - Prediction is done by forward pass through graph (forward propagation)
  - Training is done by backward pass through graph (back propagation)
  - Based on simple matrix vector operations

- Forms the basis of neural network libraries
  - Tensorflow, Pytorch, mxnet, etc.

Neural Networks

- Powerful non-linear models for classification
- Predictions are made as a sequence of simple operations
  - matrix-vector operations
  - non-linear activation functions
- Choices in network structure
  - Width and depth
  - Choice of activation function
- Feedforward networks (no loop)
- Training with the back-propagation algorithm
  - Requires defining a loss/error function
  - Gradient descent + chain rule
  - Easy to implement on top of computation graphs