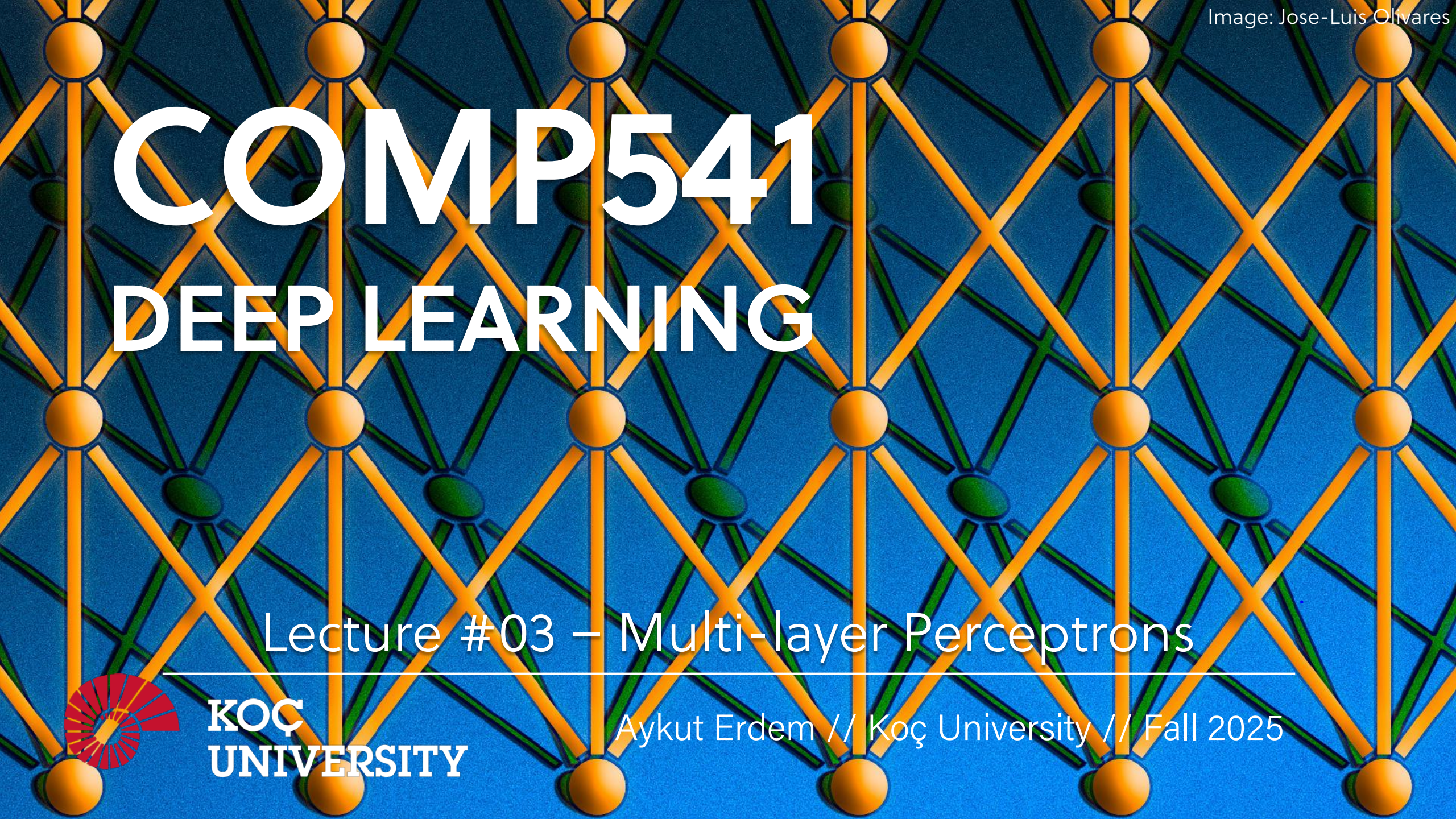


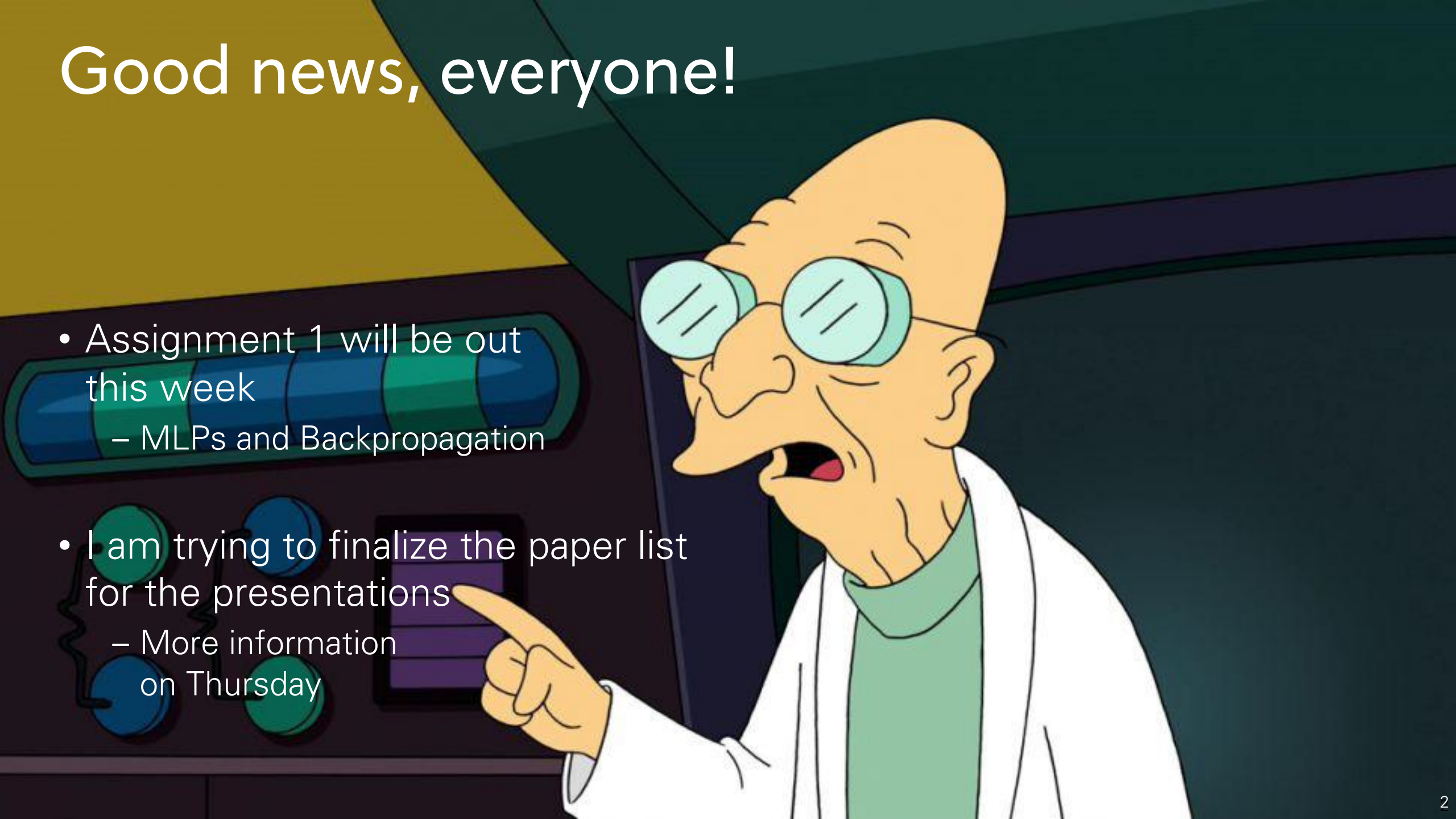
COMP541
DEEP LEARNING

Lecture #03 – Multi-layer Perceptrons

KOÇ UNIVERSITY

Aykut Erdem // Koç University // Fall 2025

Good news, everyone!



- Assignment 1 will be out this week
  - MLPs and Backpropagation
- I am trying to finalize the paper list for the presentations
  - More information on Thursday

Previously on COMP541

- learning problem
- parametric vs. non-parametric models
  - nearest-neighbor classifier
  - linear classification
  - linear regression
- capacity
- hyperparameter
- underfitting and overfitting
- variance-bias tradeoff
- model selection
- cross-validation

Lecture overview

- the perceptron
- the multi-layer perceptron
- stochastic gradient descent
- backpropagation
- shallow yet very powerful: `word2vec`

- **Disclaimer:** Much of the material and slides for this lecture were borrowed from
  —Hugo Larochelle's Neural networks slides
  —Nick Locascio's MIT 6.S191 slides
  —Efstratios Gavves and Max Willing's UvA deep learning class
  —Leonid Sigal's CPSC532L class
  —Richard Socher's CS224d class
  —Dan Jurafsky's CS124 class

A Brief History of Neural Networks

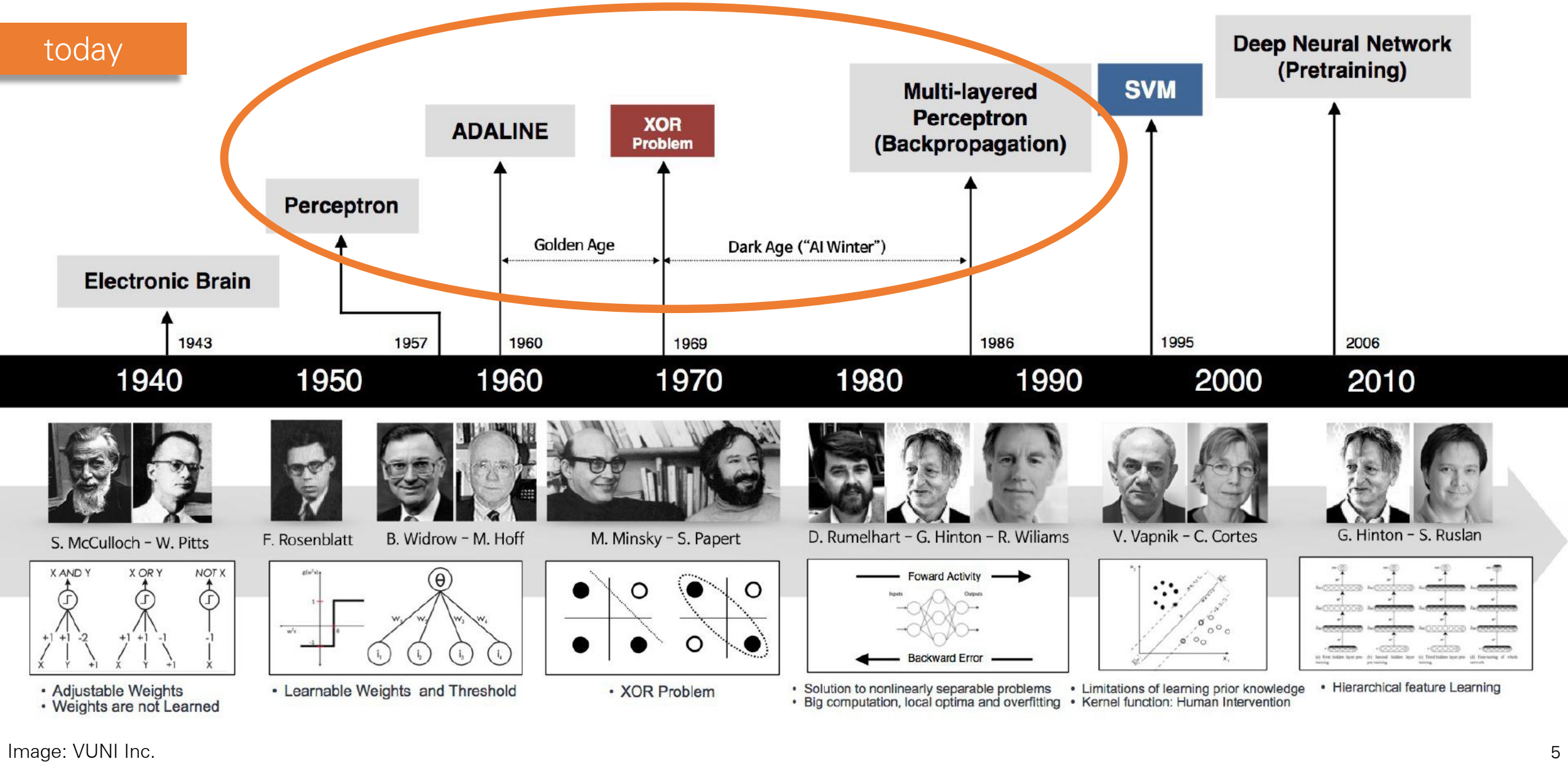
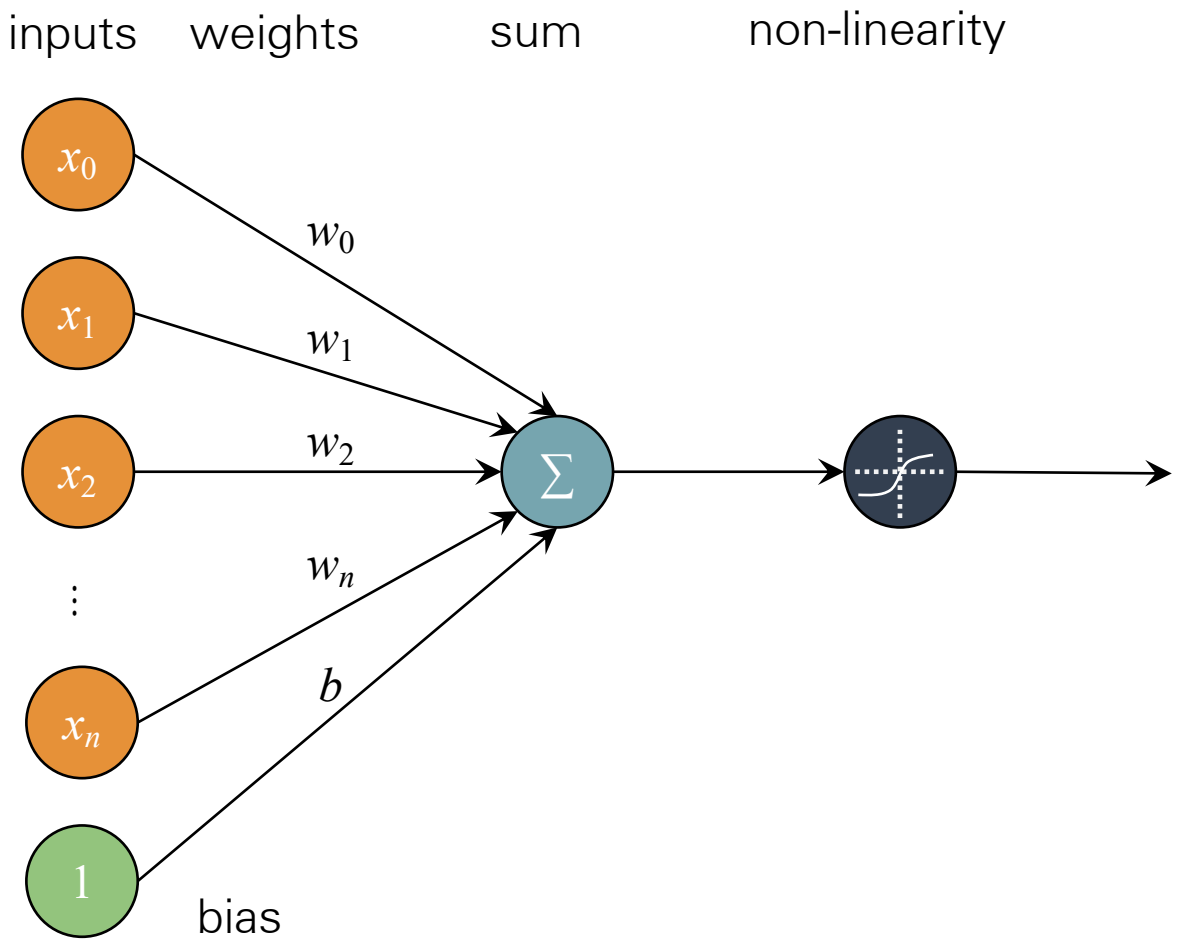


Image: VUNI Inc.

The Perceptron

The Perceptron



Perceptron Forward Pass

- Neuron pre-activation (or input activation)

$$a(\mathbf{x}) = b + \sum_i w_i x_i = b + \mathbf{w}^\top \mathbf{x}$$

- Neuron output activation:

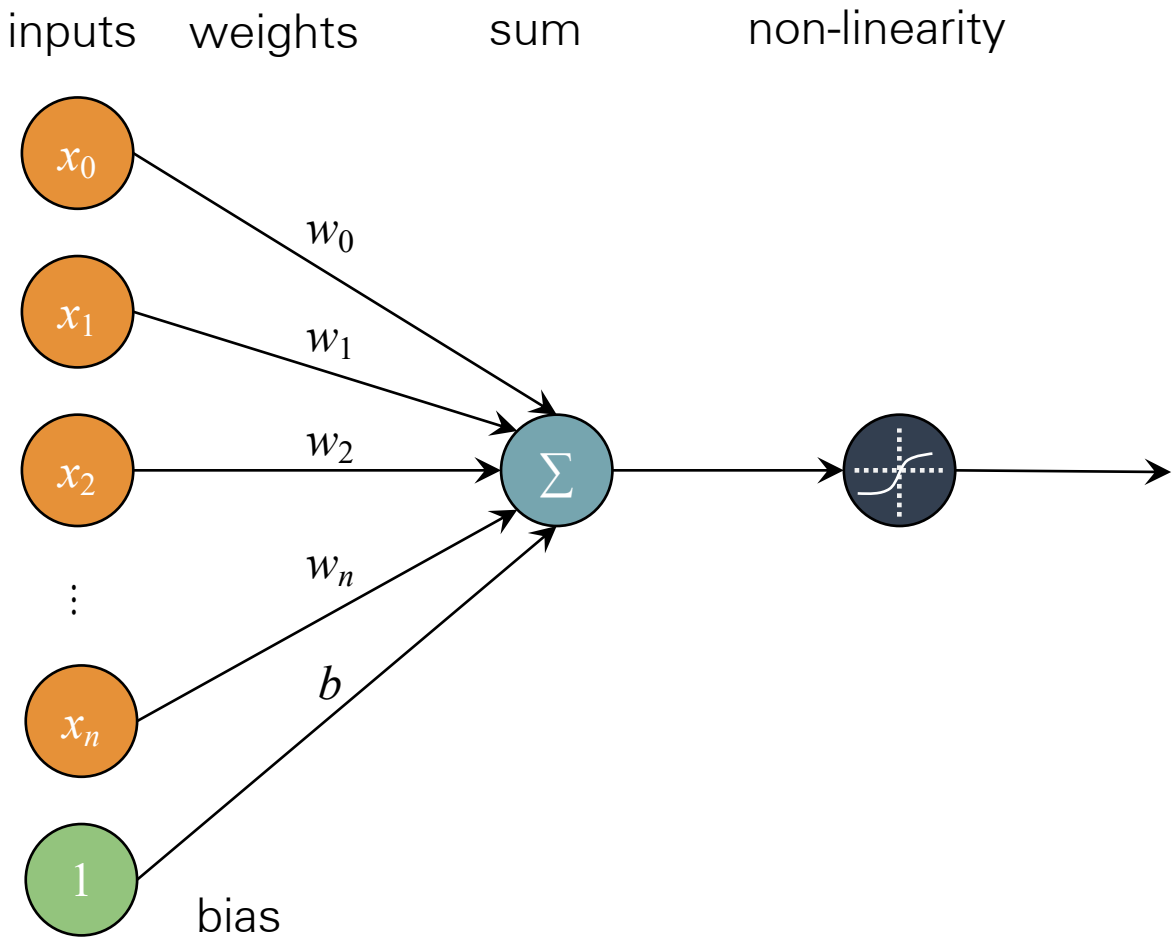
$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

where

$\mathbf{w}$ are the weights (parameters)

b is the bias term

$g(\cdot)$ is called the activation function



Output Activation of The Neuron

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

Range is determined by $g(\cdot)$

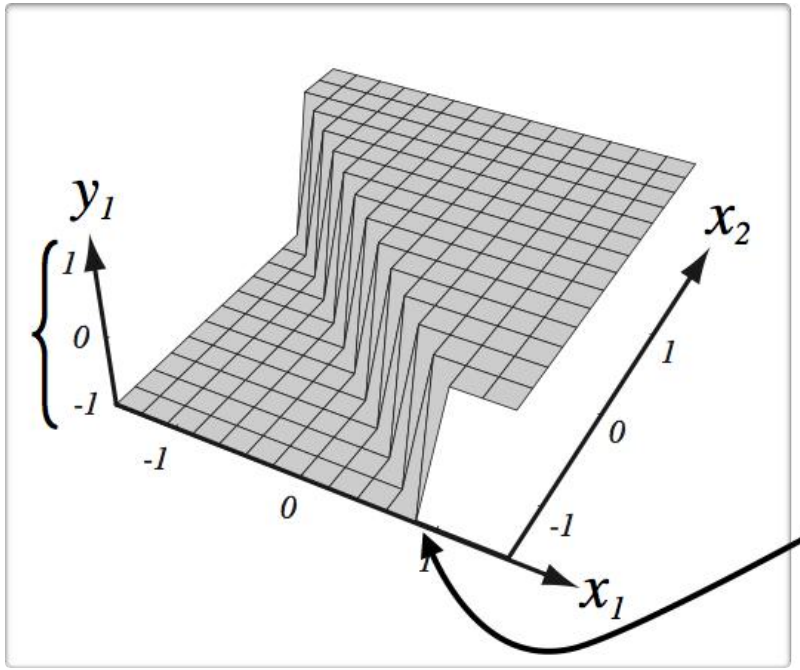
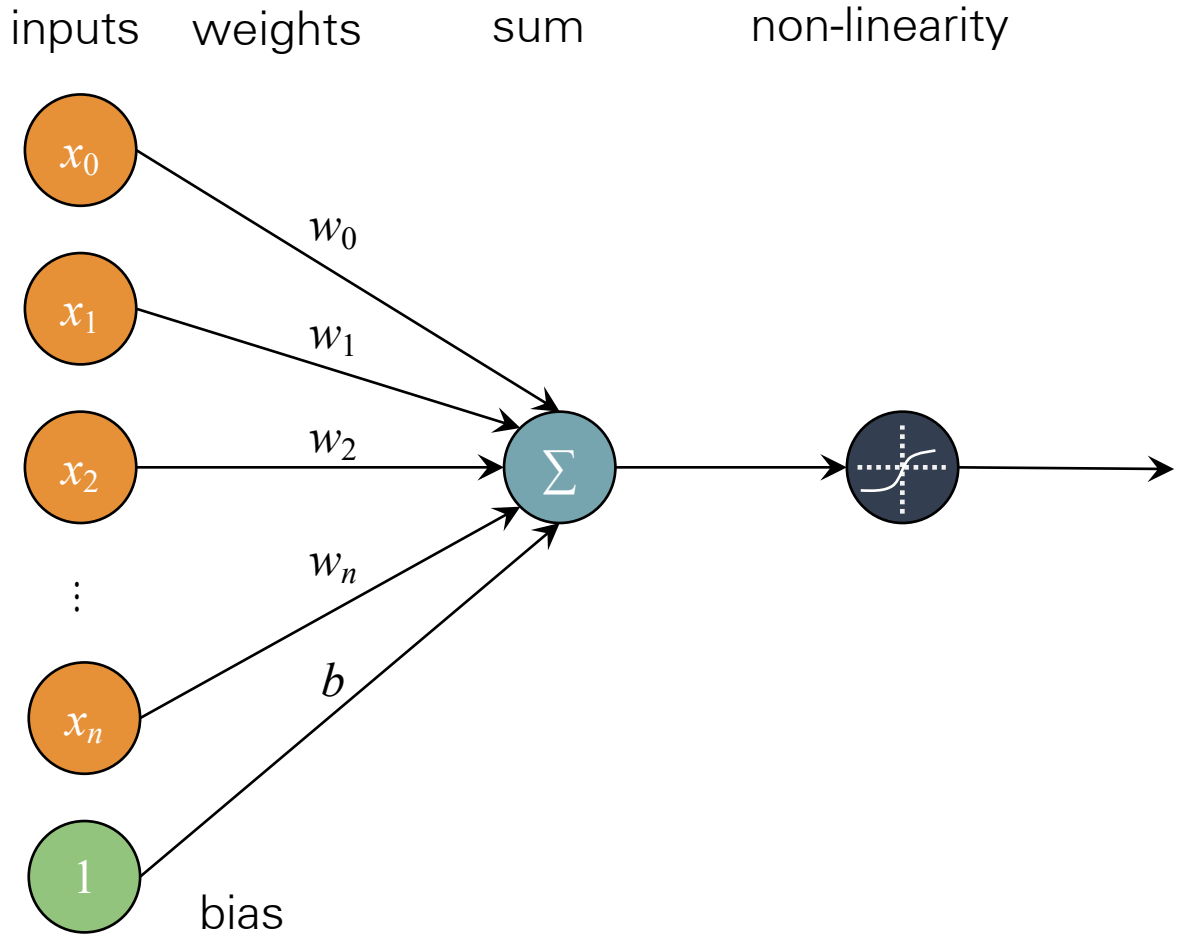


Image credit: Pascal Vincent

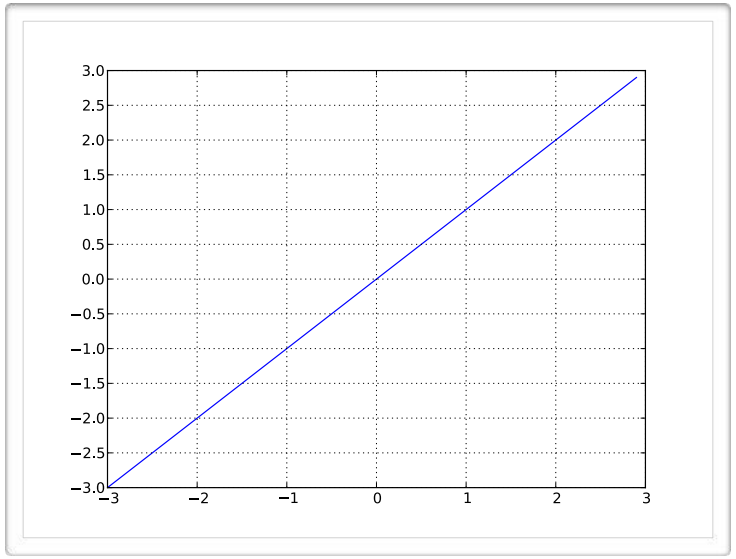
Bias only changes the position of the riff



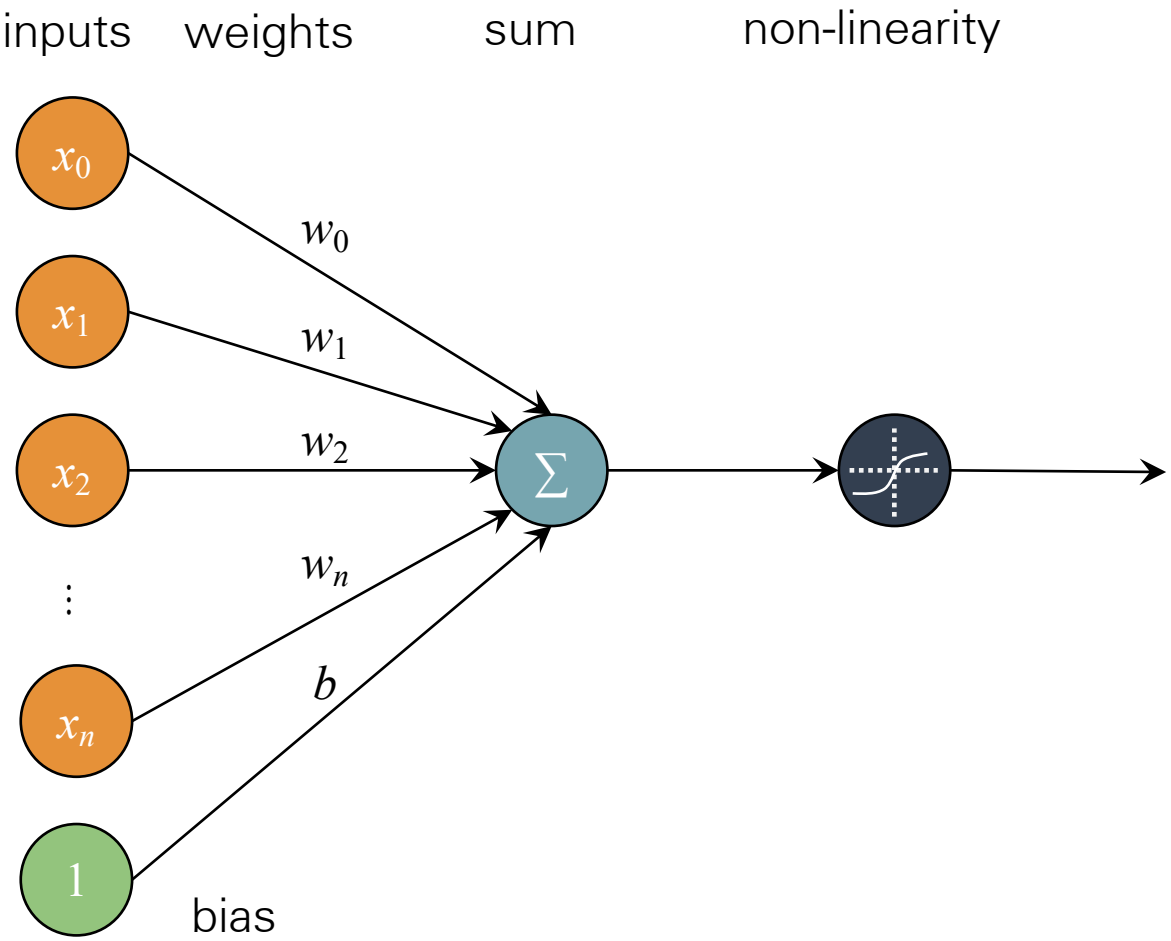
Linear Activation Function

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

$$g(a) = a$$



- No nonlinear transformation
- No input squashing

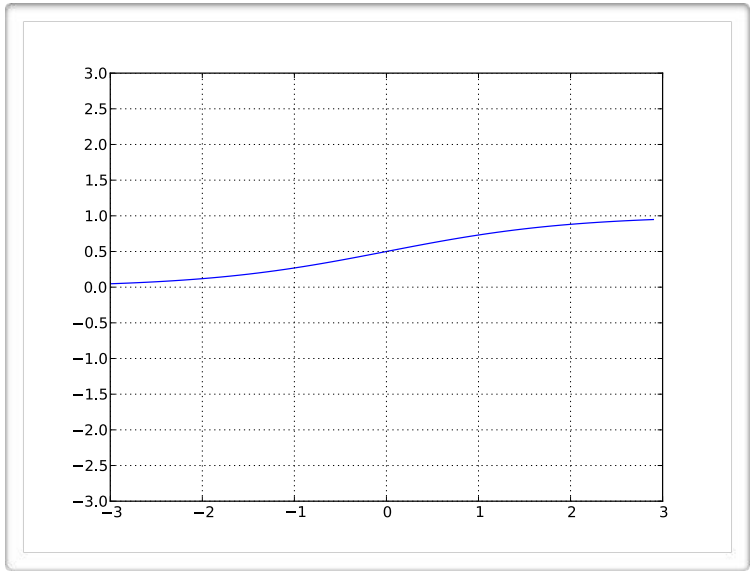


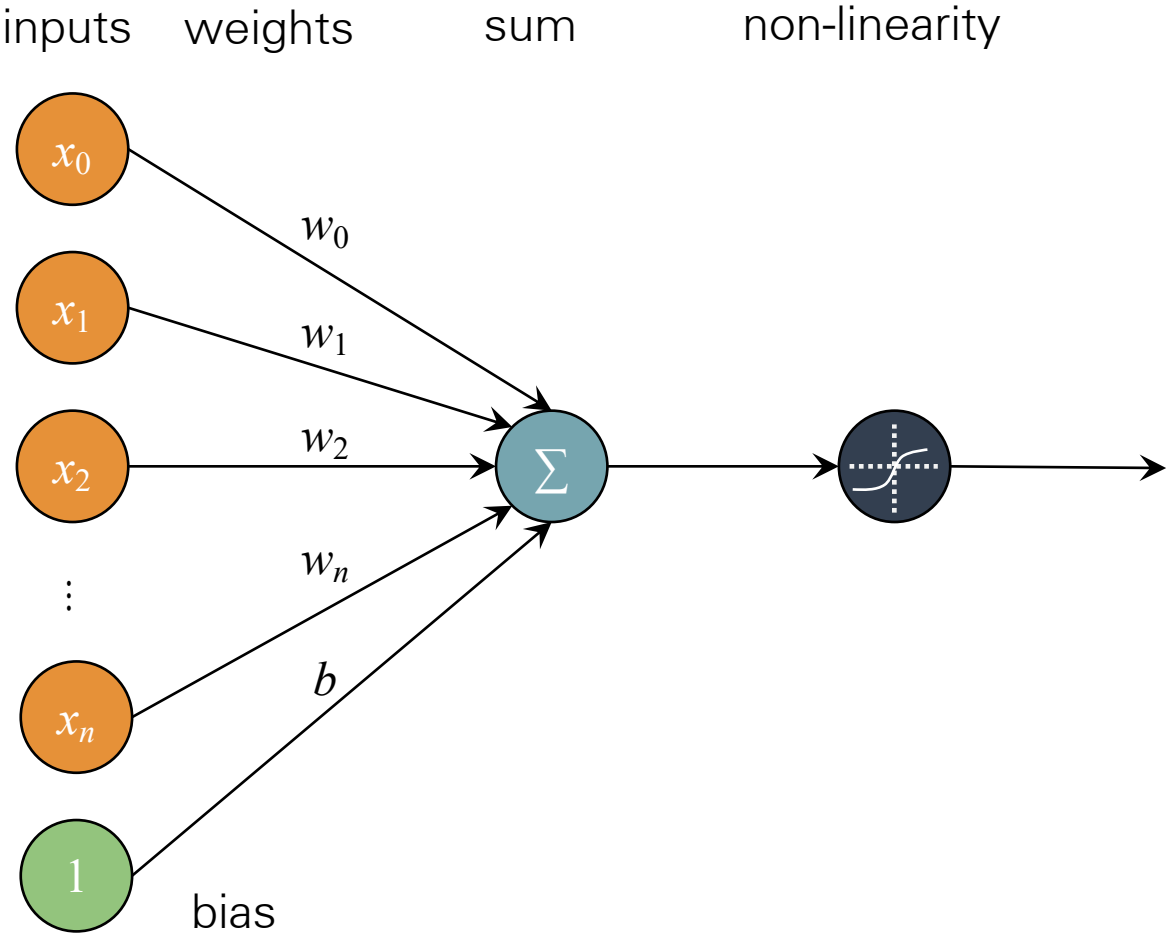
Sigmoid Activation Function

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

$$g(a) = \text{sigm}(a) = \frac{1}{1+\exp(-a)}$$

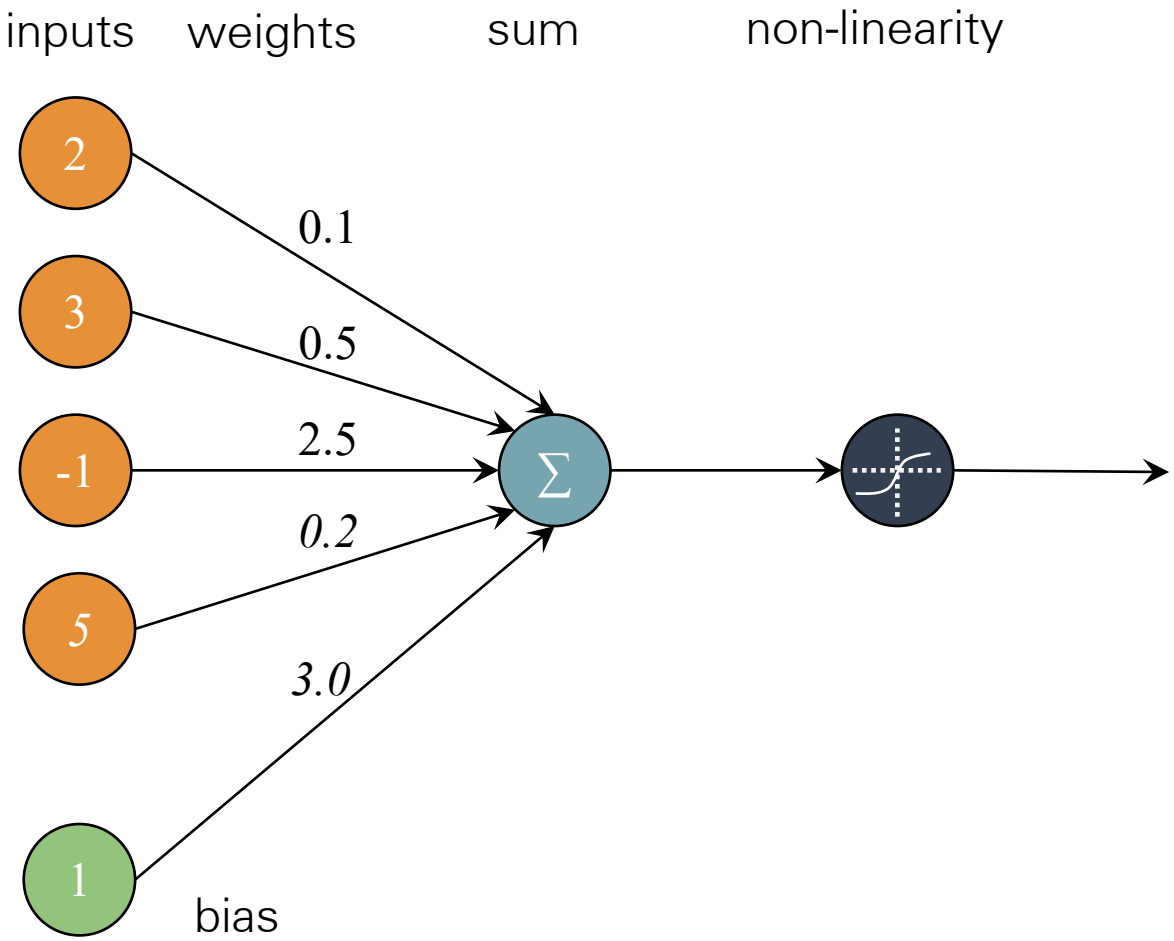
- Squashes the neuron's output between 0 and 1
- Always positive
- Bounded
- Strictly Increasing





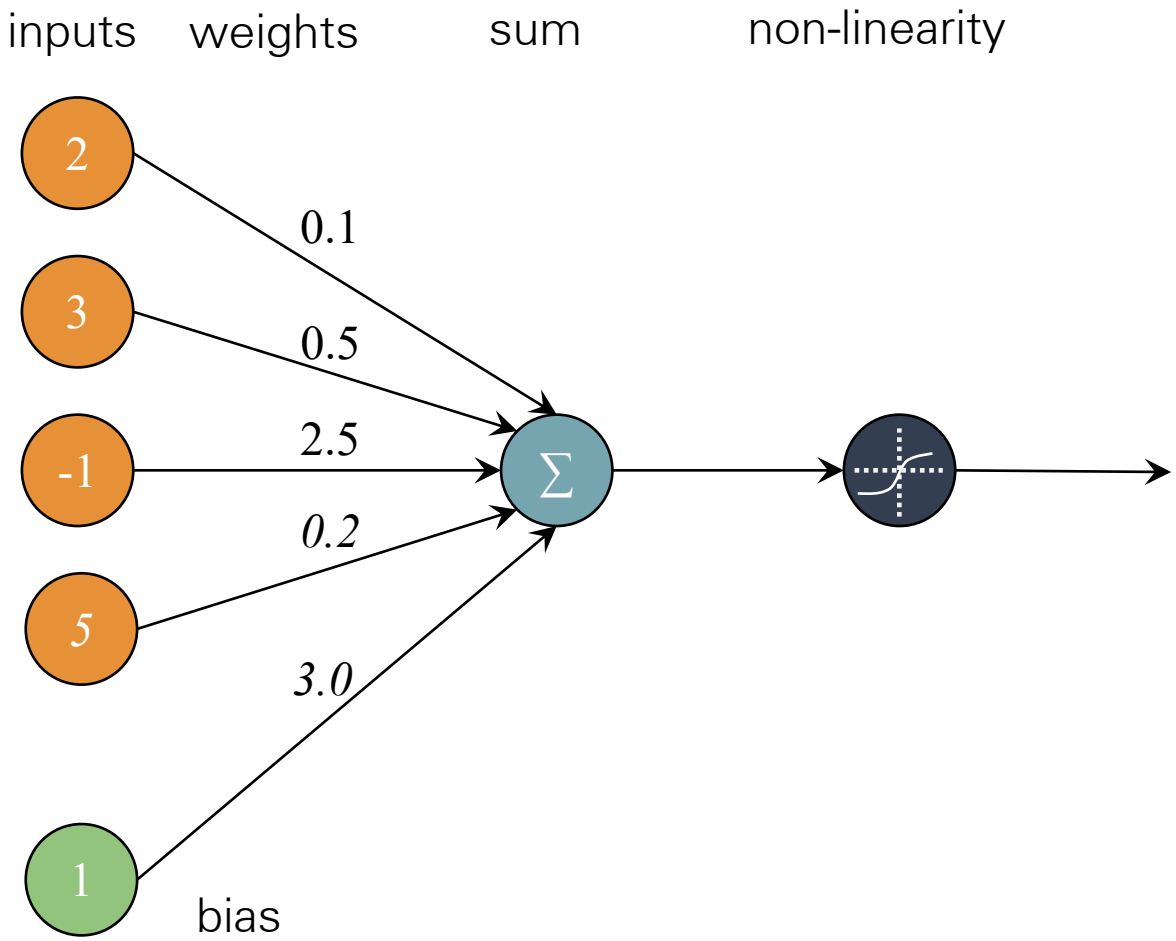
Perceptron Forward Pass

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$



Perceptron Forward Pass

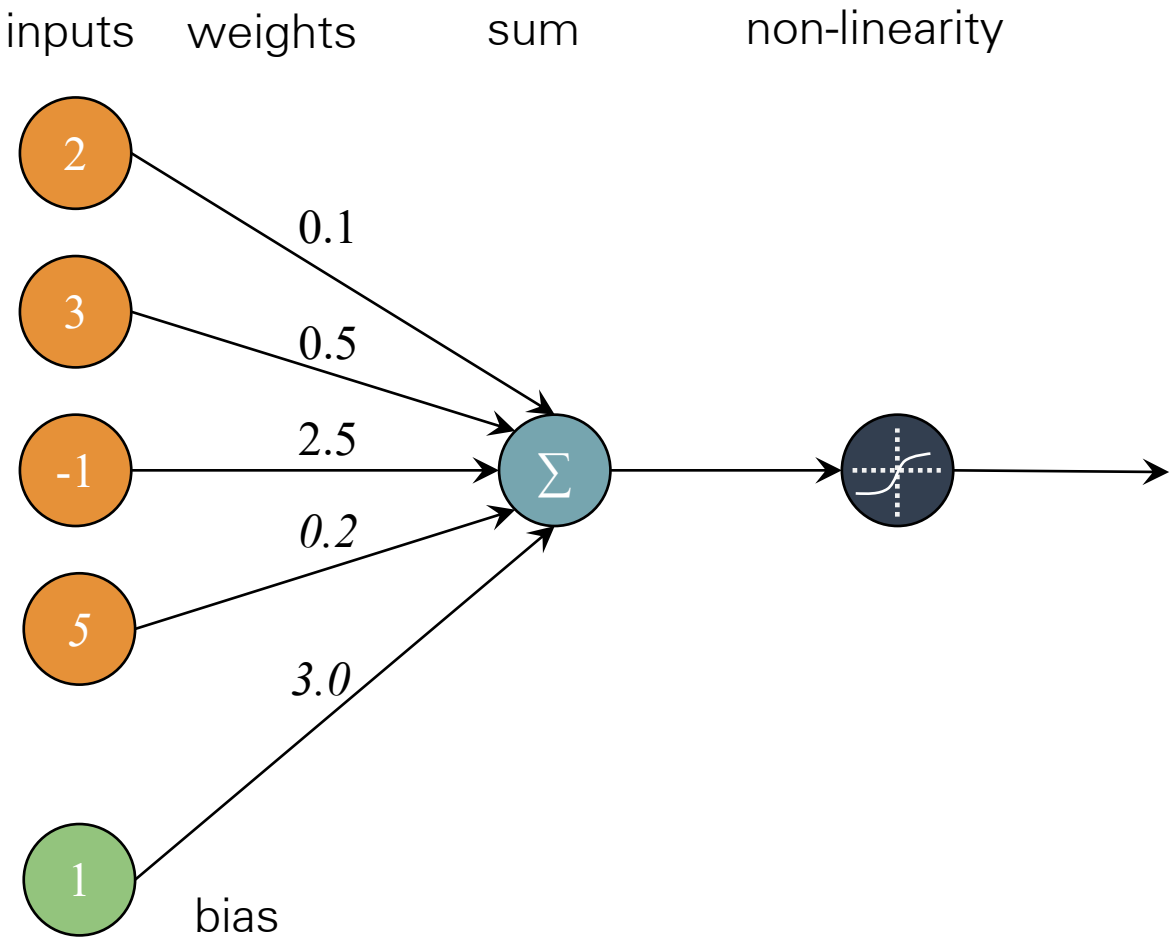
$$h(\mathbf{x}) = g\big((2*0.1) + (3*0.5) + (-1*2.5) + (5*0.2) + (1*3.0)\big)$$



Perceptron Forward Pass

$$h(\mathbf{x}) = g(3.2) = \sigma(3.2)$$

$$\frac{1}{1 + e^{-3.2}} = 0.96$$

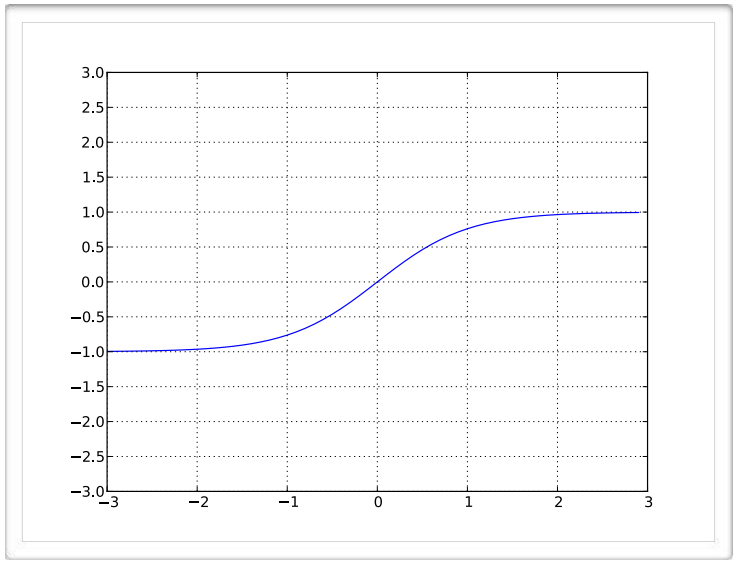


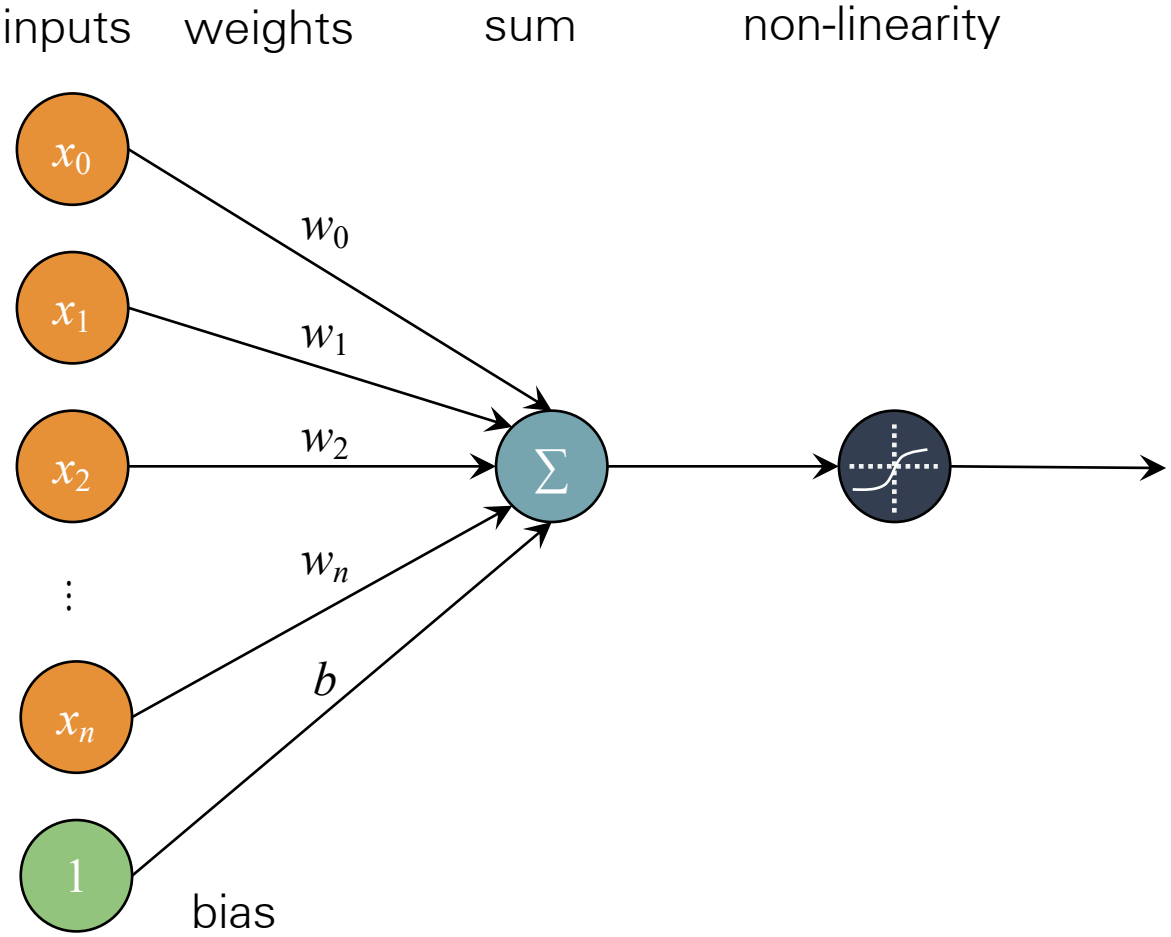
Hyperbolic Tangent (tanh) Activation Function

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

$$g(a) = \tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} = \frac{\exp(2a) - 1}{\exp(2a) + 1}$$

- Squashes the neuron's output between -1 and 1
- Can be positive or negative
- Bounded
- Strictly Increasing



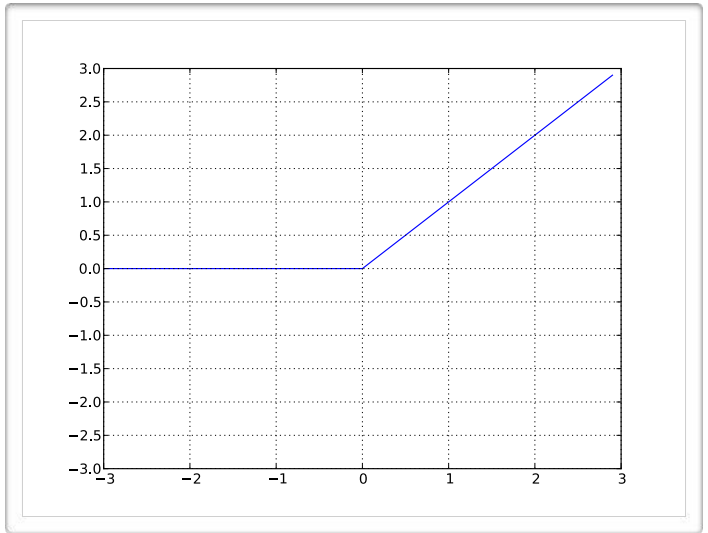


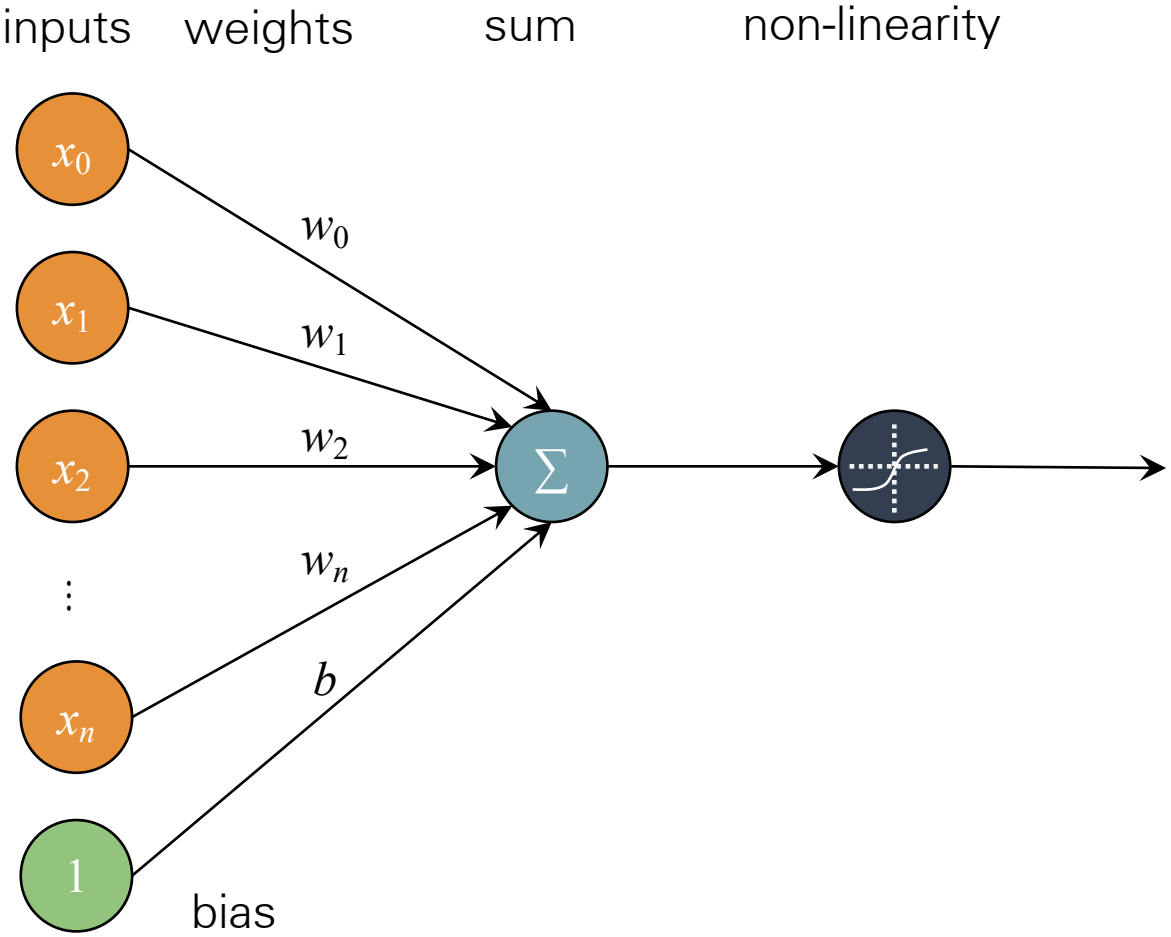
Rectified Linear (ReLU) Activation Function

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

$$g(a) = \text{reclin}(a) = \max(0, a)$$

- Bounded below by 0 (always non-negative)
- Not upper bounded
- Strictly increasing
- Tends to produce units with sparse activities





Decision Boundary of a Neuron

- Could do binary classification:
  - —with sigmoid, one can interpret neuron as estimating $p(y = 1 \mid \mathbf{x})$
  - —also known as logistic regression classifier

Decision boundary is linear

- —if activation is greater than 0.5, predict 1
- —otherwise predict 0

Same idea can be applied to a tanh activation

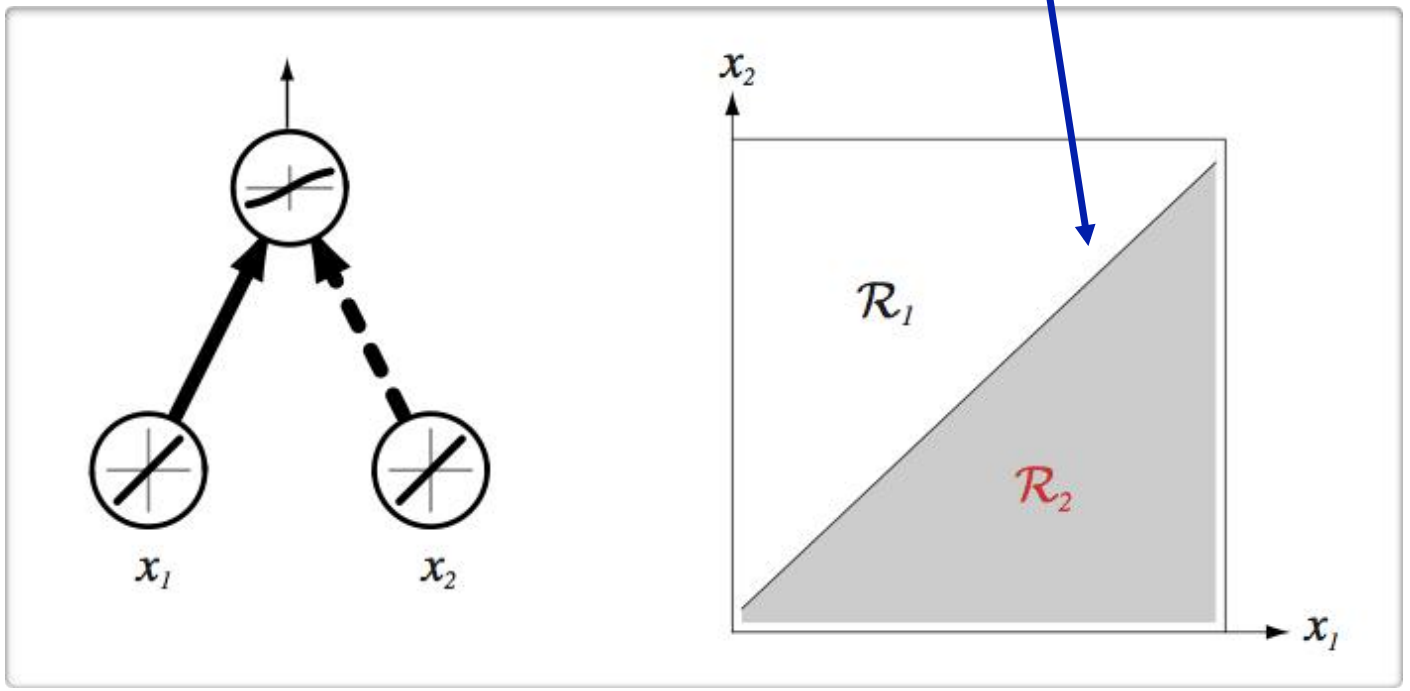
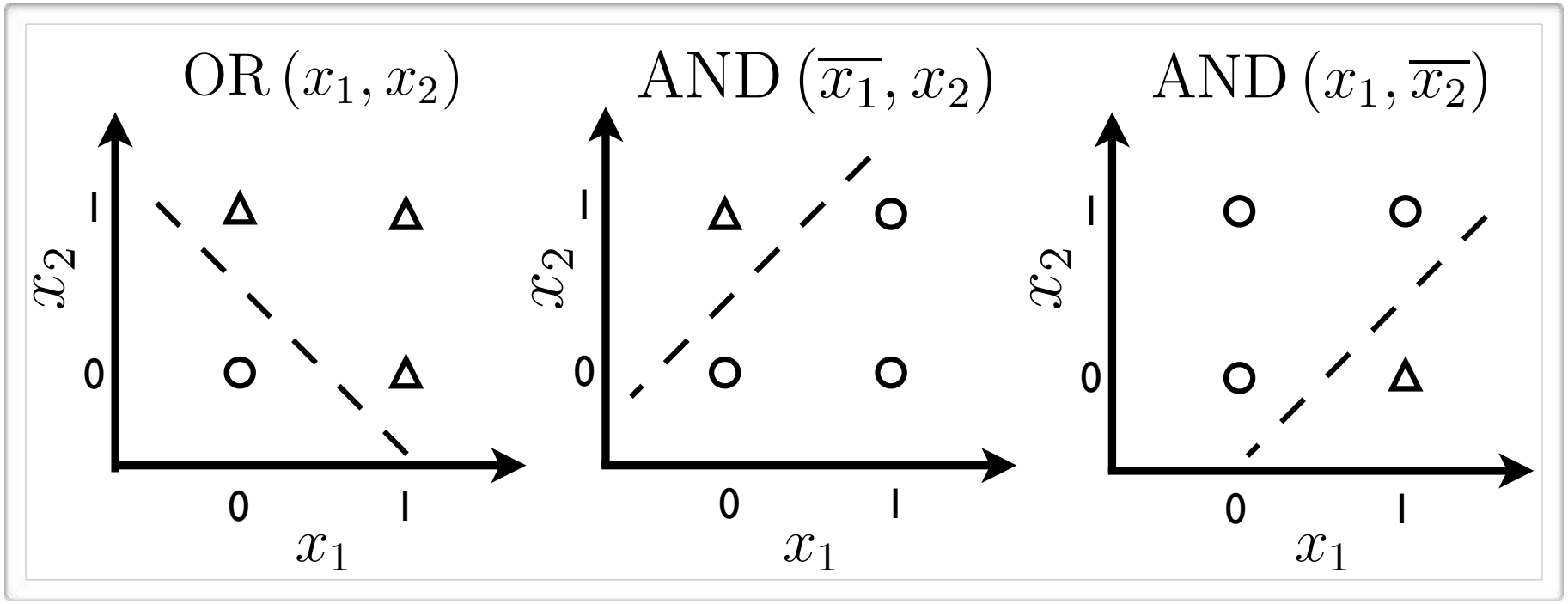


Image credit: Pascal Vincent

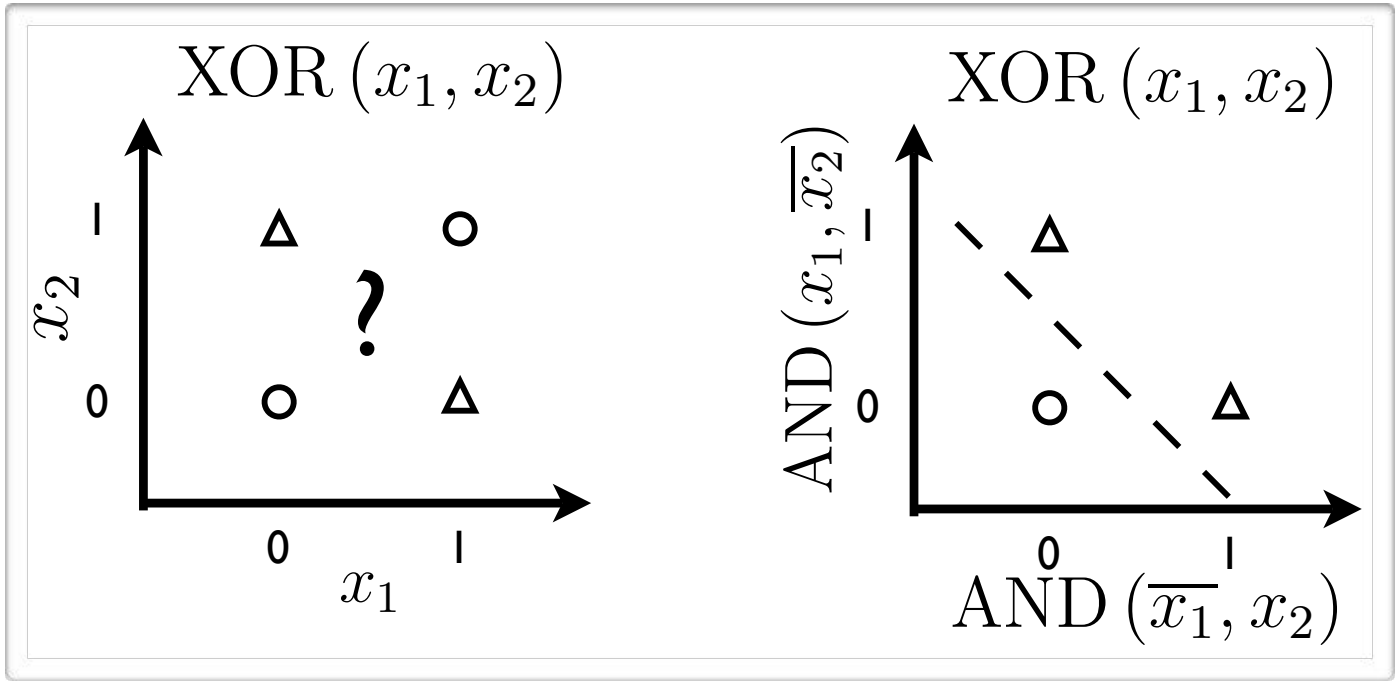
Capacity of Single Neuron

- Can solve linearly separable problems



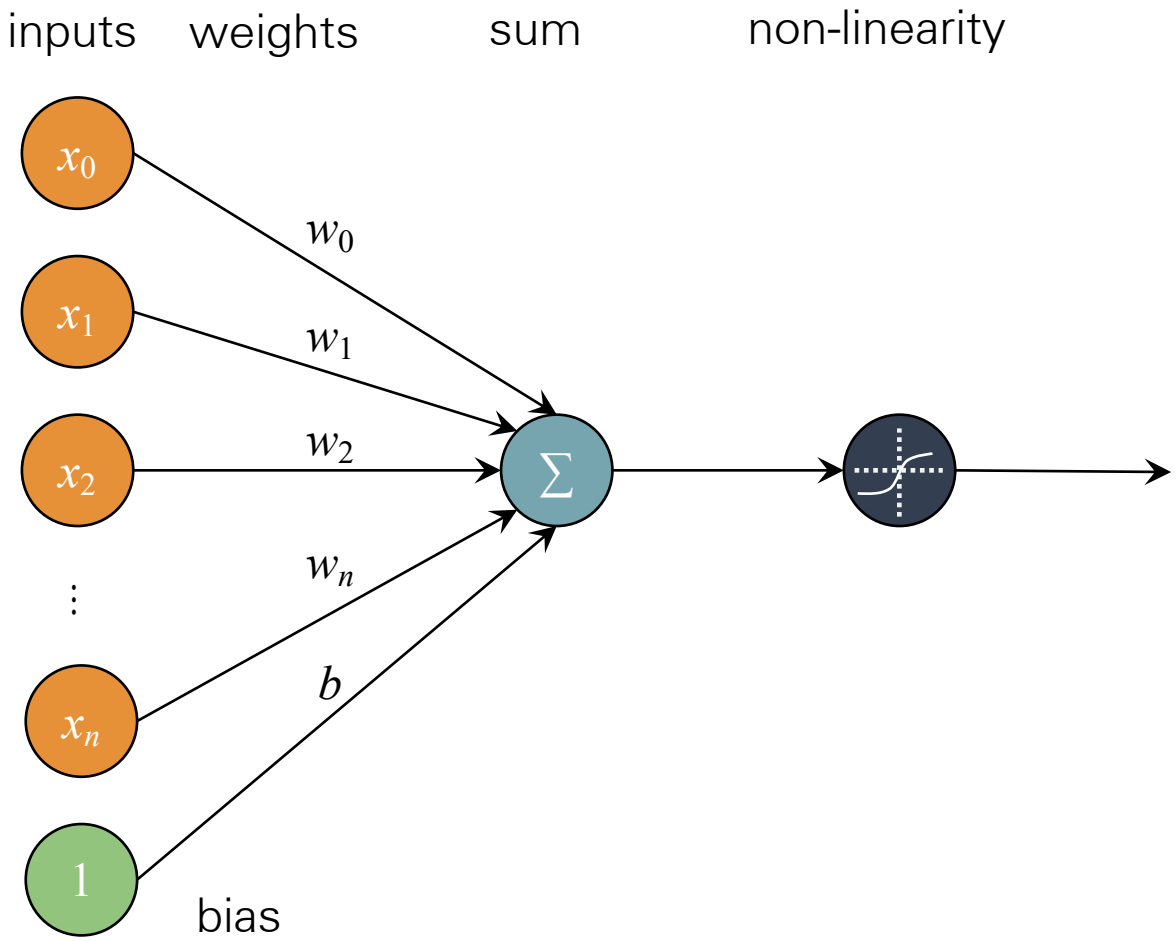
Capacity of Single Neuron

- Can not solve non-linearly separable problems

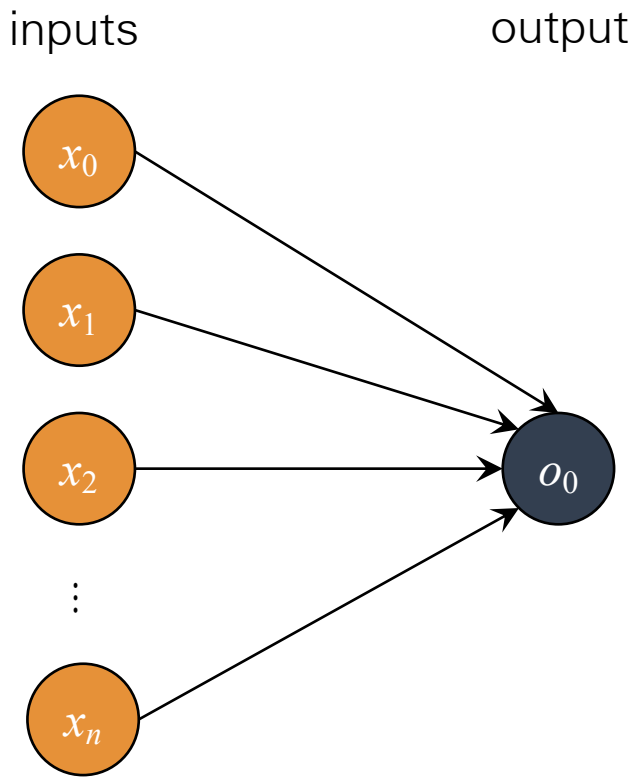


- Need to transform the input into a better representation
- Remember basis functions!

Perceptron Diagram Simplified

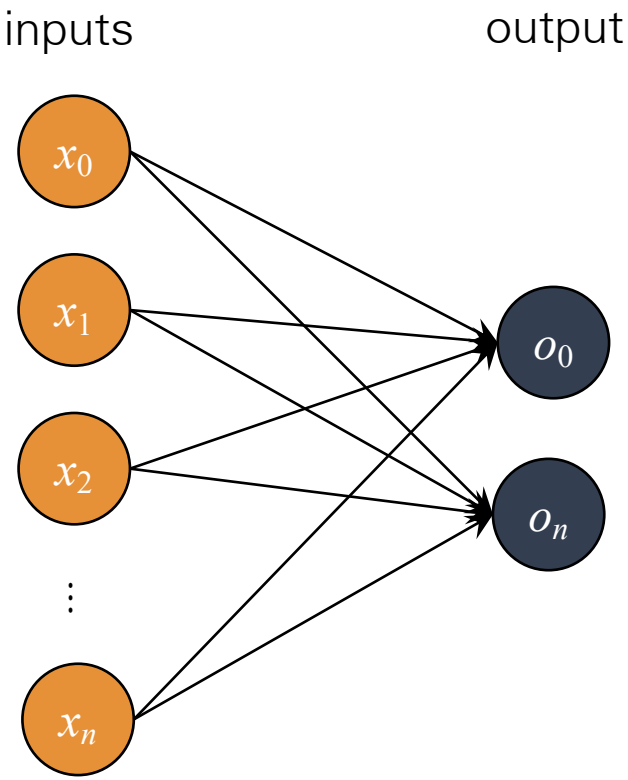


Perceptron Diagram Simplified



Multi-Output Perceptron

- Remember multi-way classification
  - —We need multiple outputs (1 output per class)
  - —We need to estimate conditional probability $p(y = \mathbf{c} \mid \mathbf{x})$
  - —Discriminative Learning



- Softmax activation function at the output

$$\mathbf{o}(\mathbf{a}) = \text{softmax}(\mathbf{a}) = \left[\frac{\exp(a_1)}{\sum_c \exp(a_c)} \cdots \frac{\exp(a_C)}{\sum_c \exp(a_c)} \right]^\top$$

  - —strictly positive
  - —sums to one

- Predict class with the highest estimated class conditional probability.

Multi-Layer Perceptron

Single Hidden Layer Neural Network

- Hidden layer pre-activation:

$$\mathbf{a}(\mathbf{x}) = \mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}$$

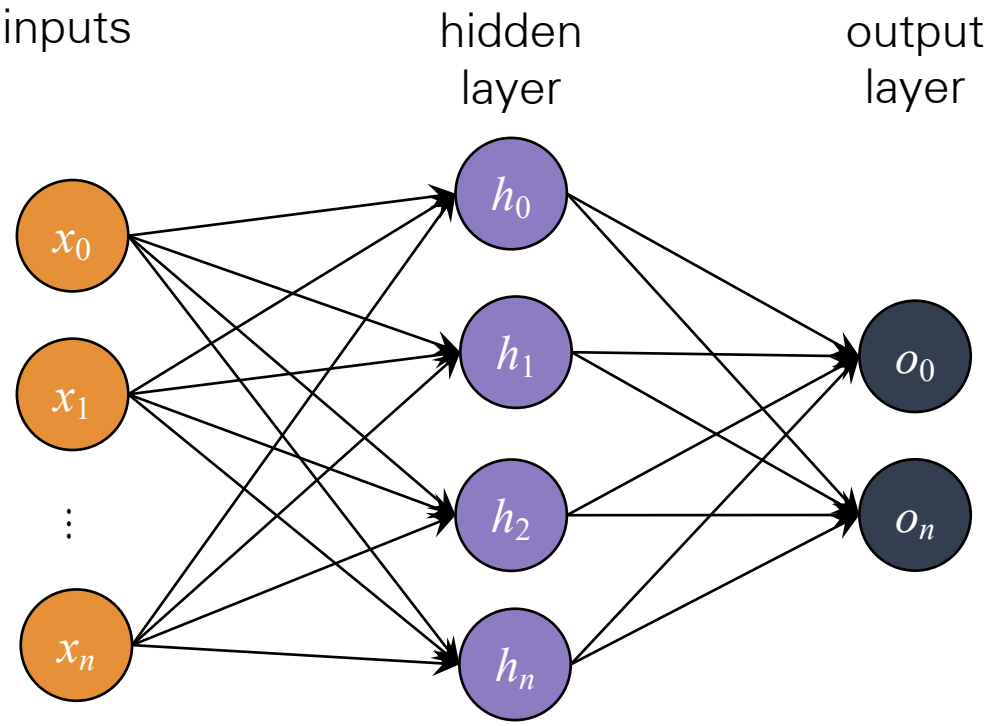
$$\left(a(\mathbf{x})_i = b_i^{(1)} + \sum_j W_{i,j}^{(1)} x_j\right)$$

- Hidden layer activation:

$$\mathbf{h}(\mathbf{x}) = \mathbf{g}(\mathbf{a}(\mathbf{x}))$$

- Output layer activation:

$$\mathbf{o}(\mathbf{x}) = \mathbf{o}\left(b^{(2)} + \mathbf{w}^{(2)\top}\mathbf{h}^{(1)}\mathbf{x}\right)$$



Multi-Layer Perceptron (MLP)

- Consider a network with L hidden layers.

—layer pre-activation for k>0

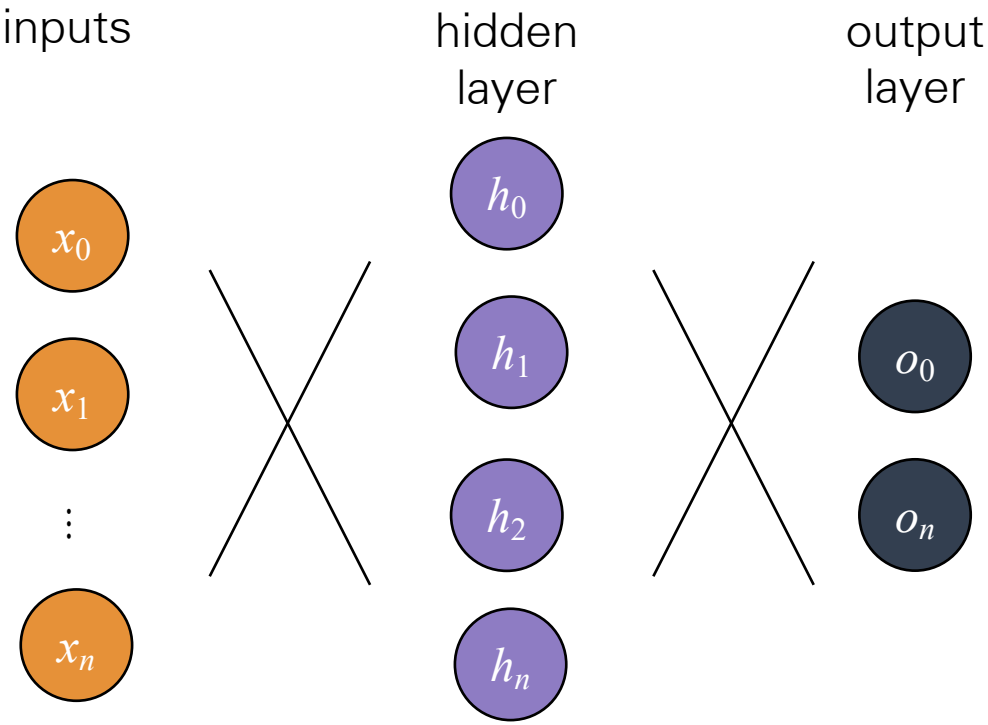
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

—hidden layer activation from 1 to L:

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

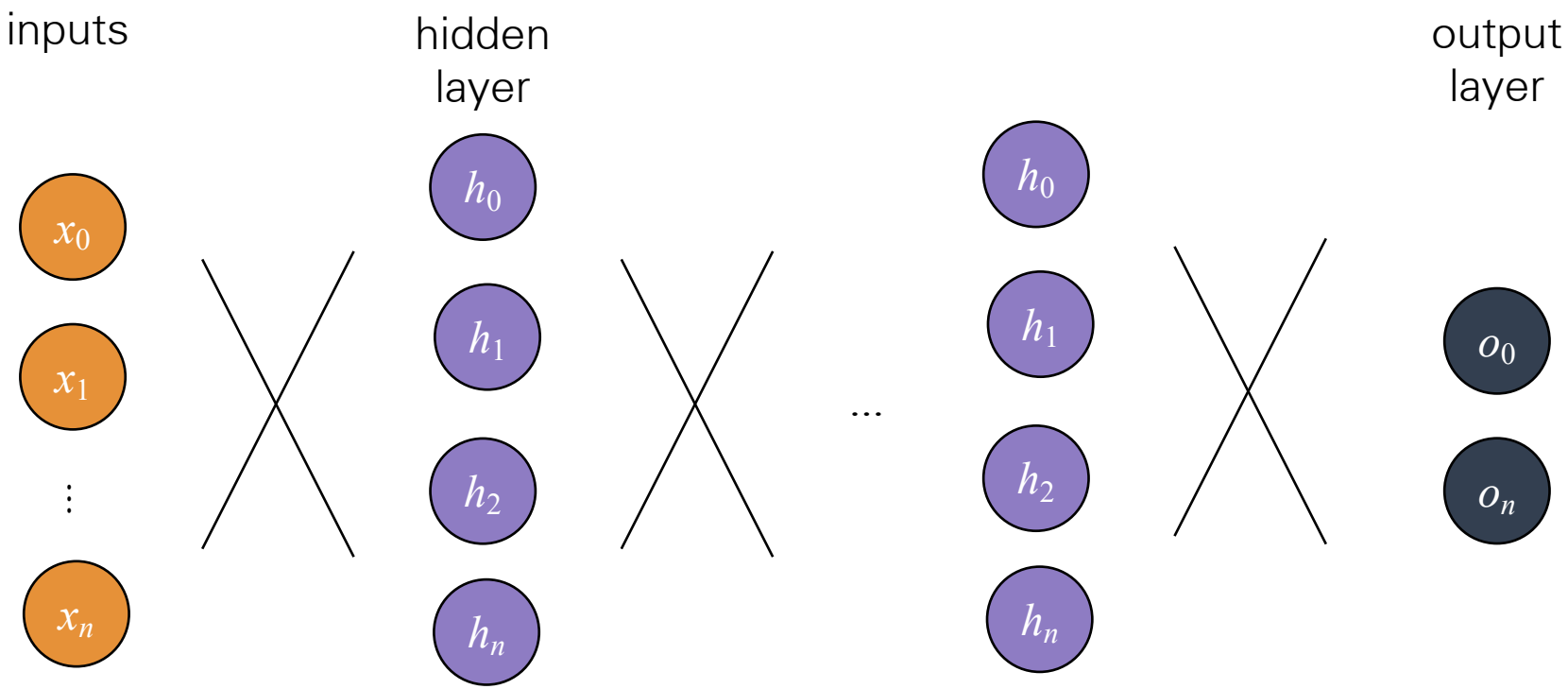
—output layer activation (k=L+1)

$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



$$(\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x})$$

Deep Neural Network



Capacity of Neural Networks

- Consider a single layer neural network

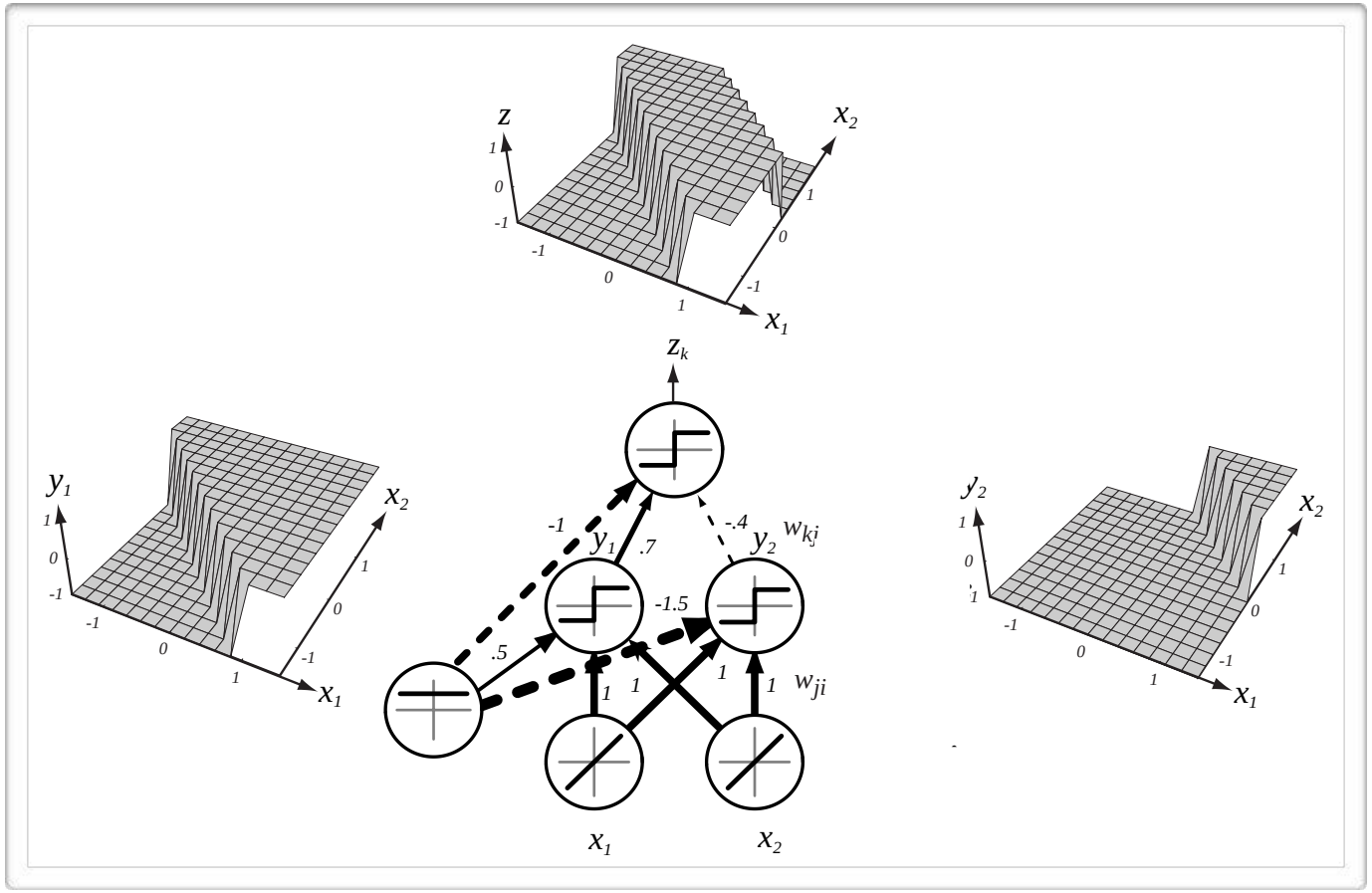


Image credit: Pascal Vincent

Capacity of Neural Networks

- Consider a single layer neural network

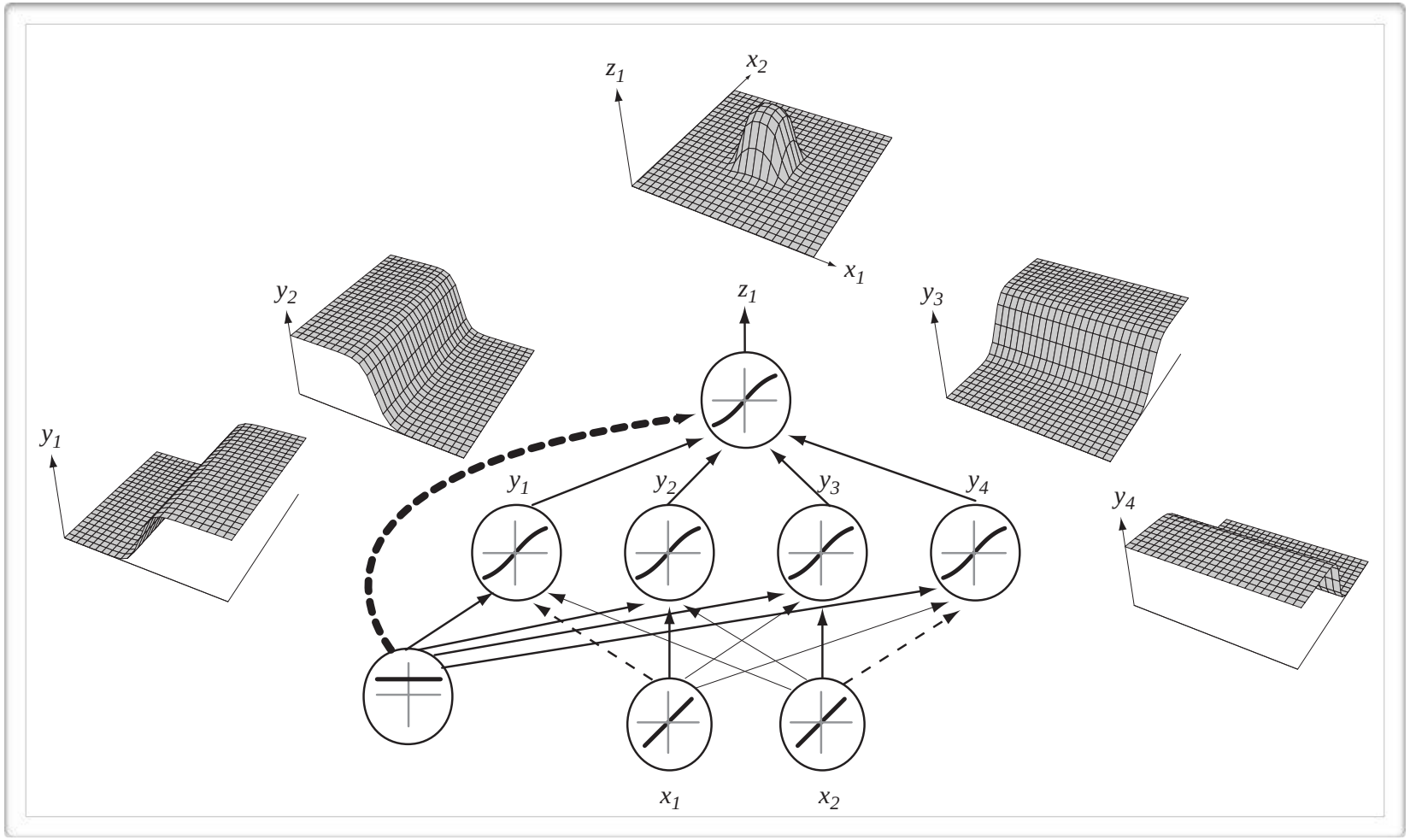


Image credit: Pascal Vincent

Capacity of Neural Networks

- Consider a single layer neural network

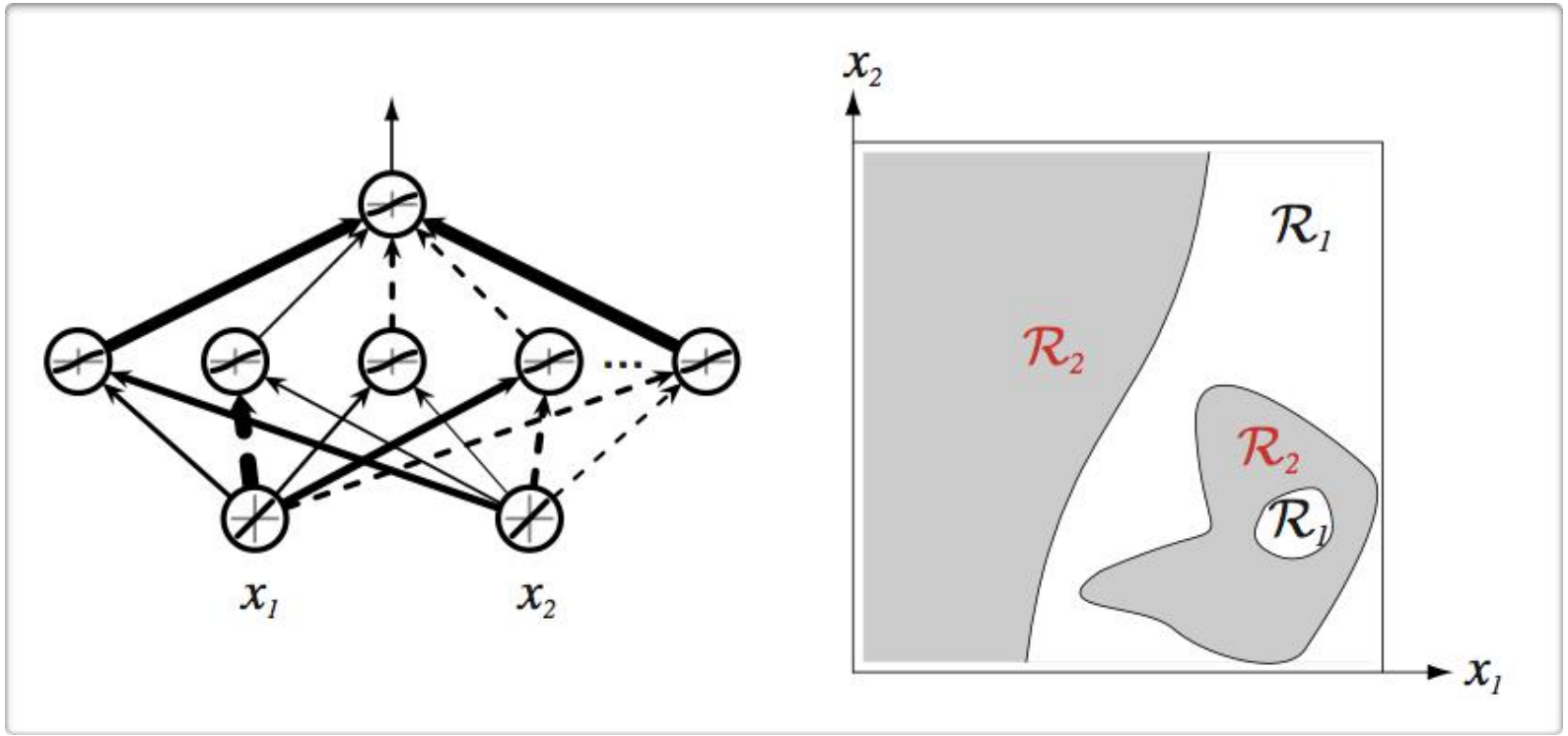


Image credit: Pascal Vincent

Universal Approximation

- Universal Approximation Theorem (Hornik, 1991):
  - —"a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units''
- This applies for sigmoid, tanh and many other activation functions.
- However, this does not mean that there is learning algorithm that can find the necessary parameter values.

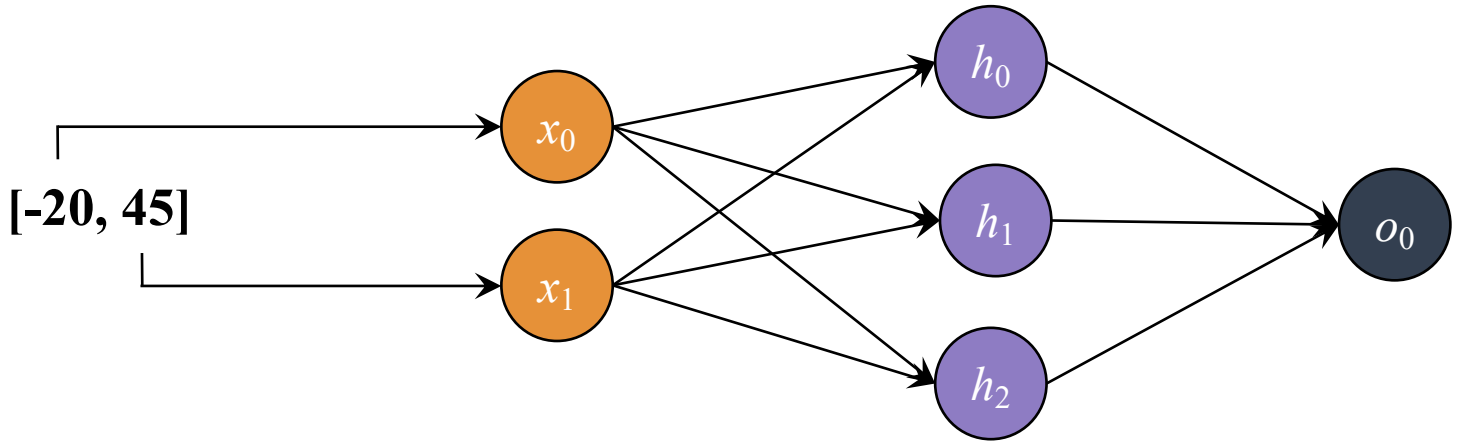
Applying Neural Networks

Example Problem: Will my flight be delayed?

Temperature: -20 F

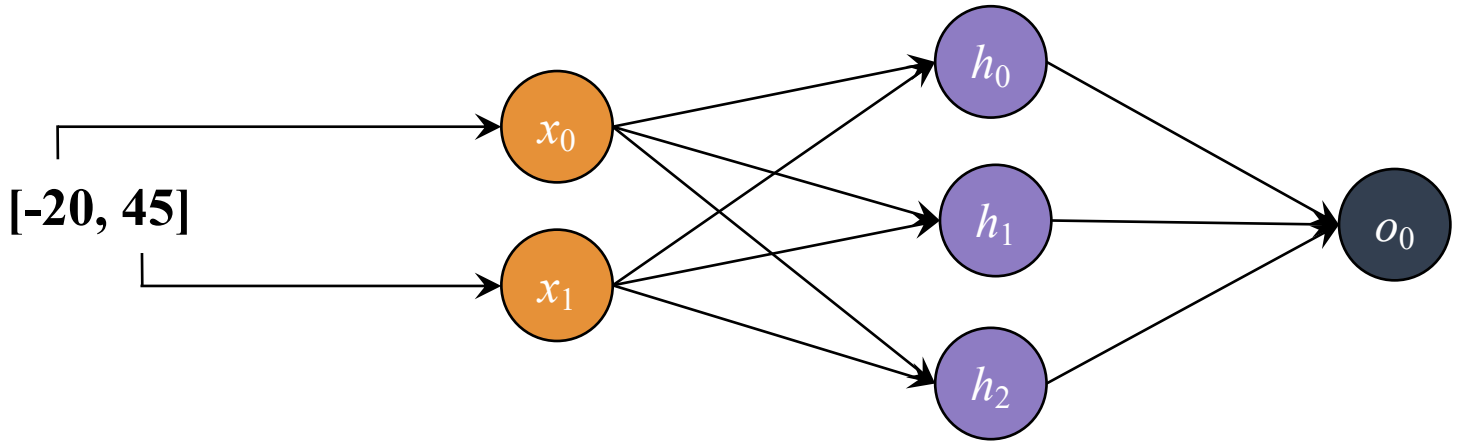
Wind Speed: 45 mph

Example Problem: Will my flight be delayed?



Predicted: 0.05

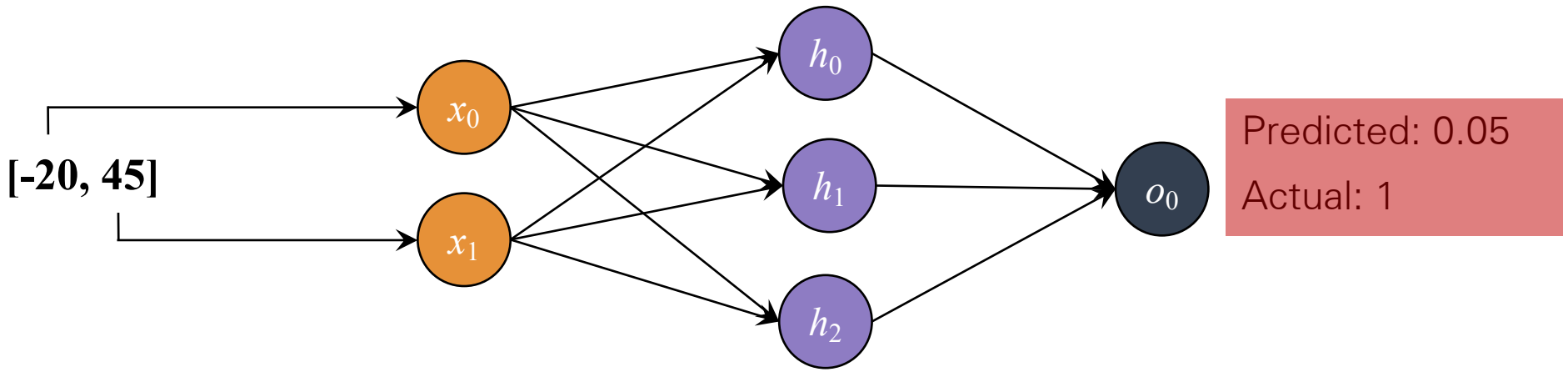
Example Problem: Will my flight be delayed?



Predicted: 0.05

Actual: 1

Quantifying Loss

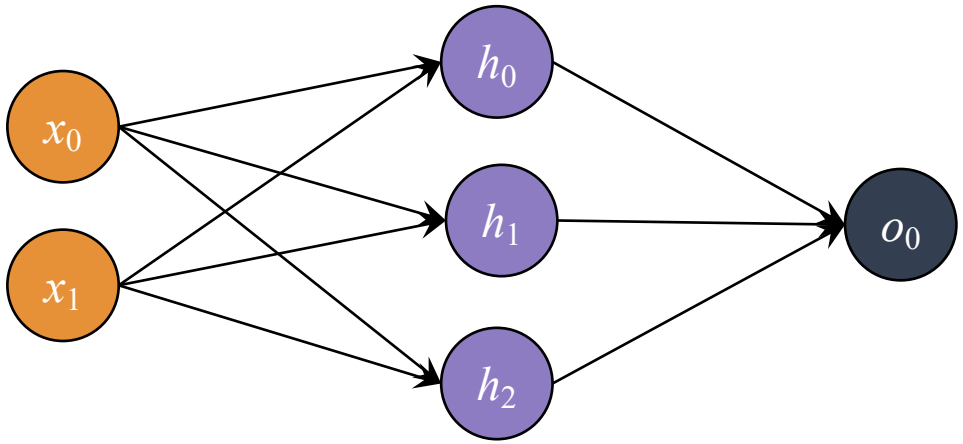


$$\ell(\underbrace{f(\mathbf{x}^{(i)};\theta)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}})$$

Total Loss

Input

[
[-20, 45],
[80, 0],
[4, 15],
[45, 60],
]



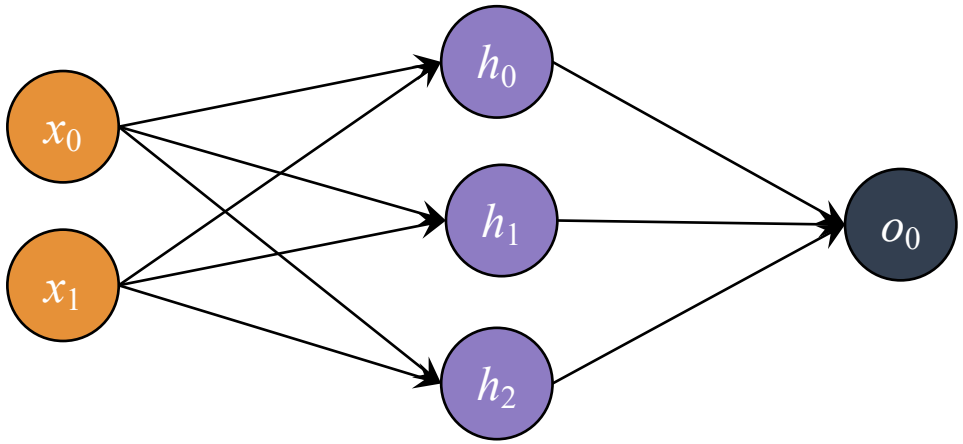
Predicted	Actual
[	[
0.05	1
0.02	0
0.96	1
0.35	1
]	]

$$J(\theta) = \frac{1}{N} \sum_i \ell(\underbrace{f(\mathbf{x}^{(i)}; \theta)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}})$$

Total Loss

Input

[
[-20, 45],
[80, 0],
[4, 15],
[45, 60],
]



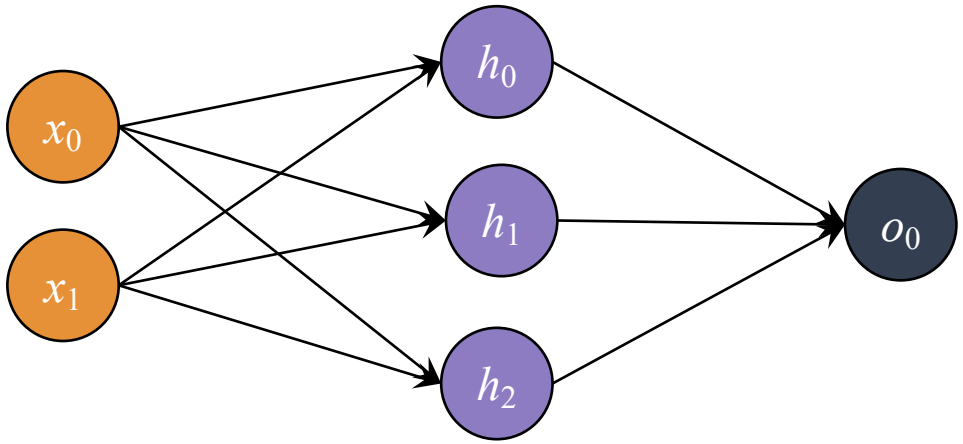
Predicted	Actual
[	[
0.05	1
0.02	0
0.96	1
0.35	1
]	]

$$J(\theta) = \frac{1}{N} \sum_i \ell(\underbrace{f(\mathbf{x}^{(i)}; \theta)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}})$$

Binary Cross Entropy Loss

Input

[
[-20, 45],
[80, 0],
[4, 15],
[45, 60],
]



Predicted	Actual
[	[
0.05	1
0.02	0
0.96	1
0.35	1
]	]

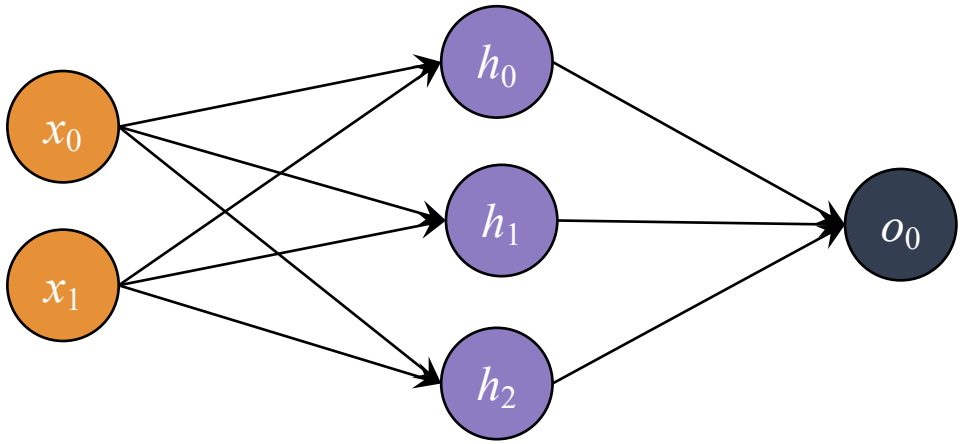
$$J_{\text{cross\_entropy}}(\theta) = \frac{1}{N} \sum_i y^{(i)} \log(f(\mathbf{x}^{(i)}; \theta)) + (1 - y^{(i)}) \log(1 - f(\mathbf{x}^{(i)}; \theta)))$$

- For classification problems with a softmax output layer.
- Maximize log-probability of the correct class given an input

Binary Cross Entropy Loss

Input

[
[-20, 45],
[80, 0],
[4, 15],
[45, 60],
]



Predicted	Actual
[	[
0.05	1
0.02	0
0.96	1
0.35	1
]	]

$$J_{\mathrm{MSE}}(\theta) = \frac{1}{N} \sum_{i} \left(f(\mathbf{x}^{(i)}; \theta) - y^{(i)} \right)^2$$

Training Neural Networks

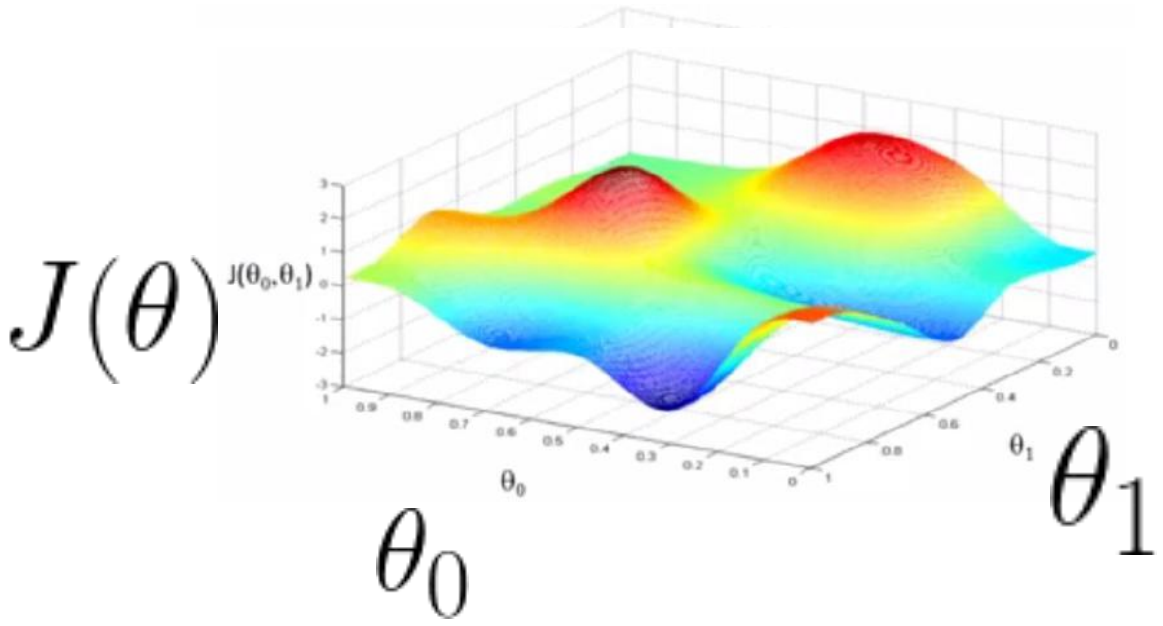
Training

$$J(\theta) = \arg\min_{\boldsymbol{\theta}} \frac{1}{T} \sum_t \underbrace{l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})}_{\text{Loss function}} + \underbrace{\lambda \Omega(\boldsymbol{\theta})}_{\text{Regularizer}}$$

$$\theta = W_1, W_2 \ldots W_n$$

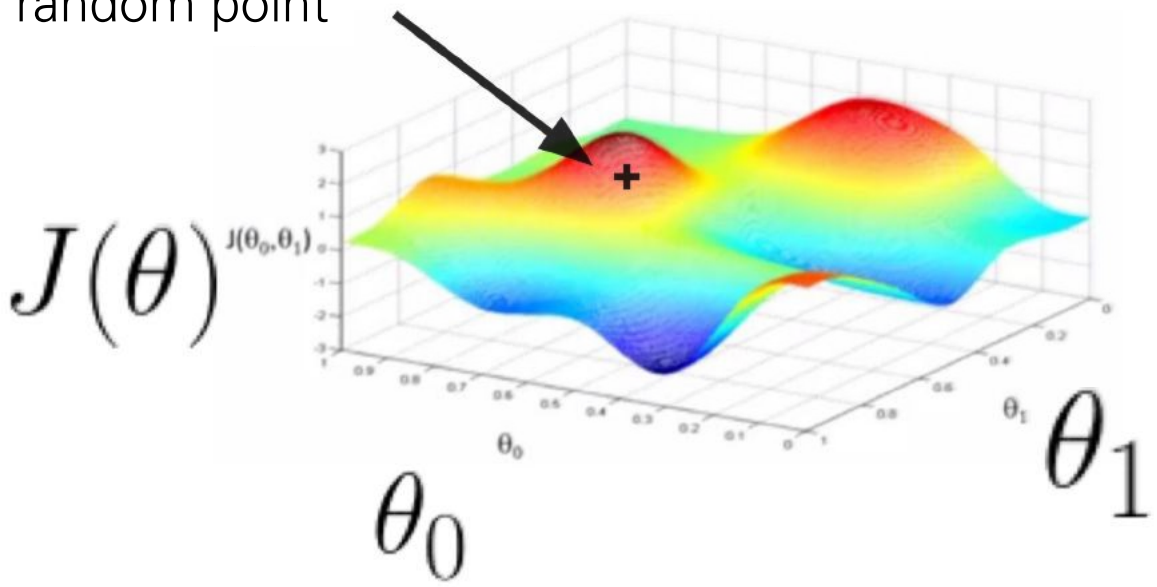
- Learning is cast as optimization.
  - —For classification problems, we would like to minimize classification error
  - —Loss function can sometimes be viewed as a surrogate for what we want to optimize (e.g. upper bound)

Loss is a **function** of the model's parameters



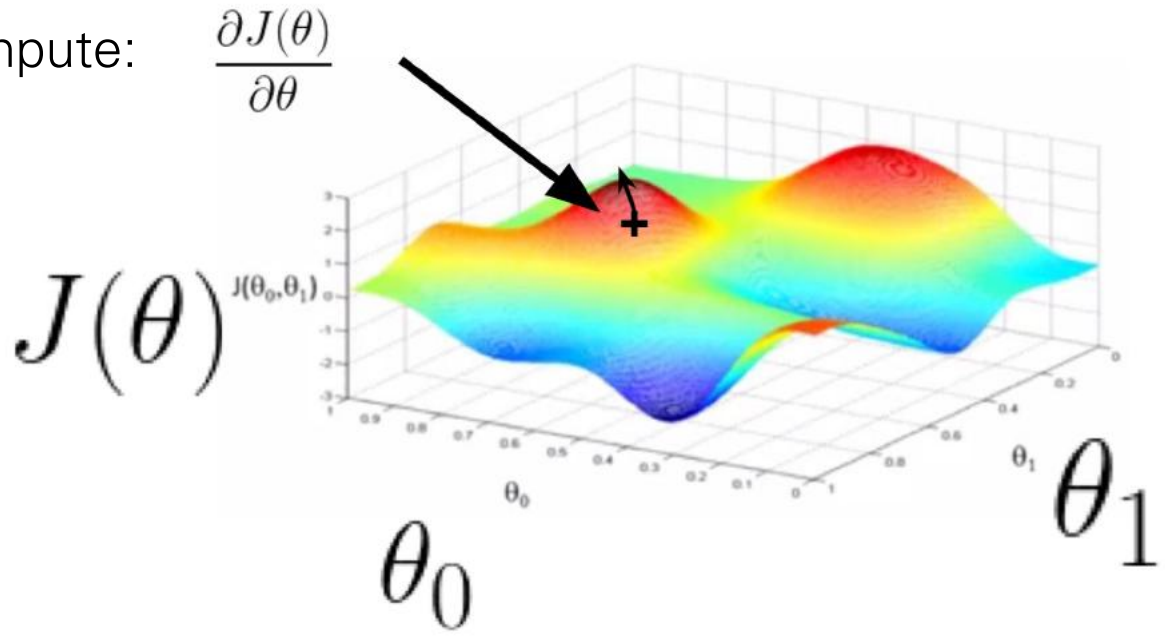
How to minimize loss?

Start at random point



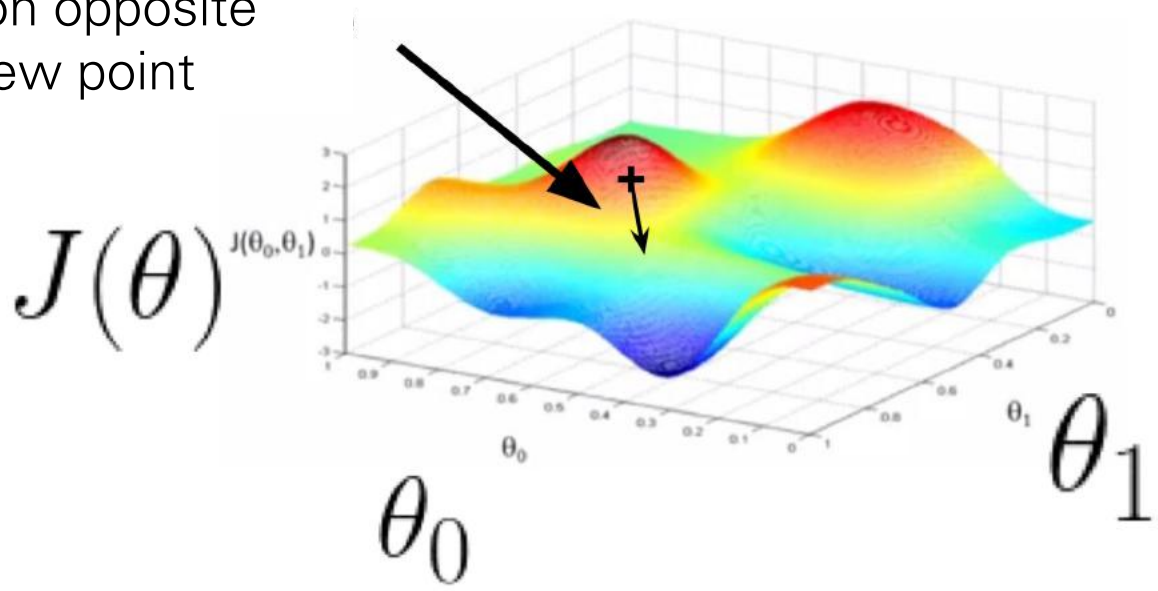
How to minimize loss?

Compute: $\frac{\partial J(\theta)}{\partial \theta}$



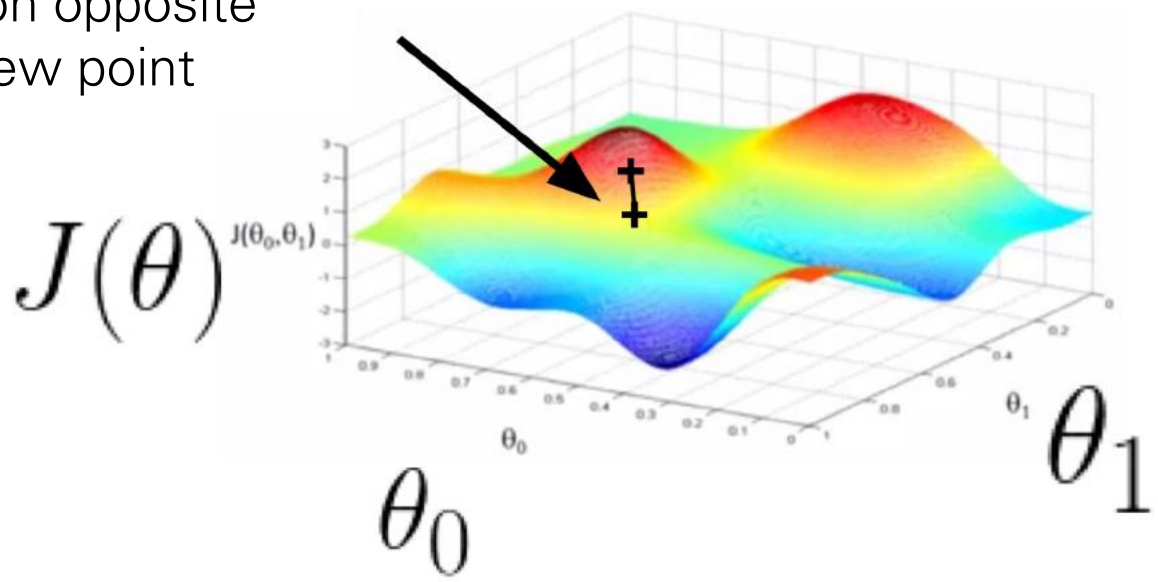
How to minimize loss?

Move in direction opposite of gradient to new point



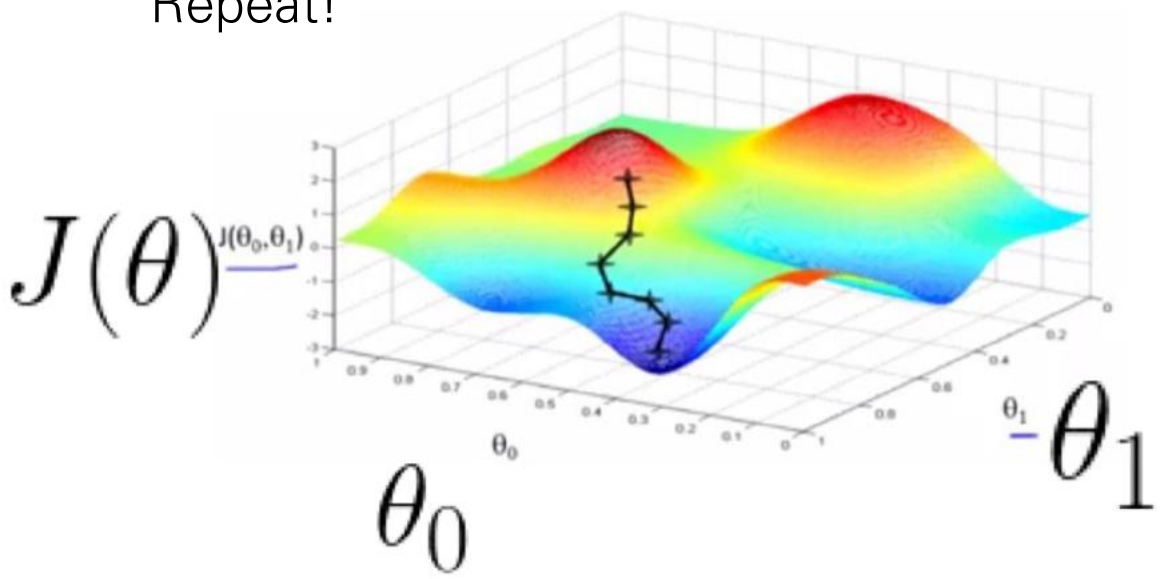
How to minimize loss?

Move in direction opposite of gradient to new point



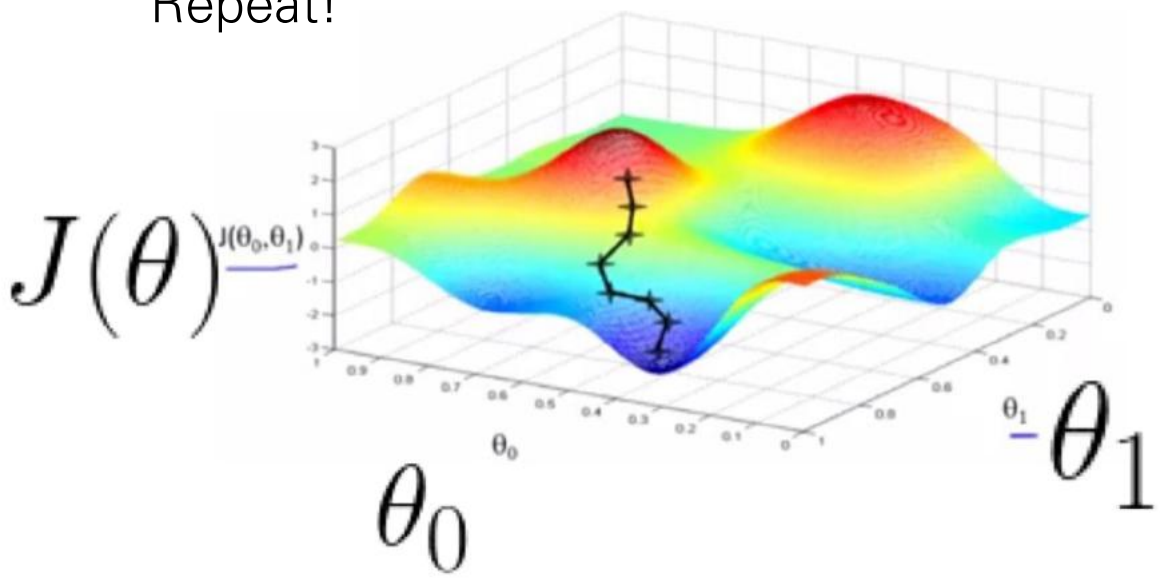
How to minimize loss?

Repeat!



This is called Stochastic Gradient Descent (SGD)

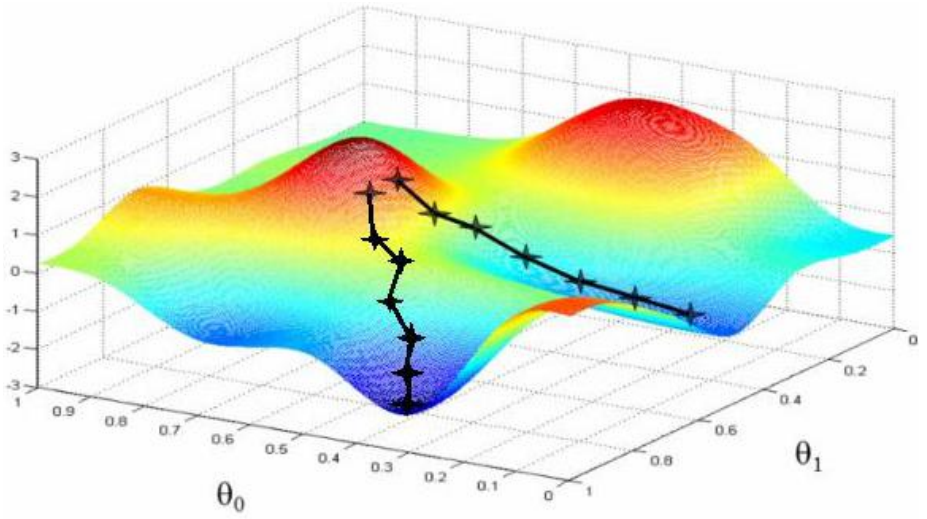
Repeat!



Stochastic Gradient Descent (SGD)

- Initialize θ randomly
- For N Epochs
  - For each training example (x, y):
    - Compute Loss Gradient: $\dfrac{\partial J(\theta)}{\partial \theta}$
    - Update θ with update rule:

$$\theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$$

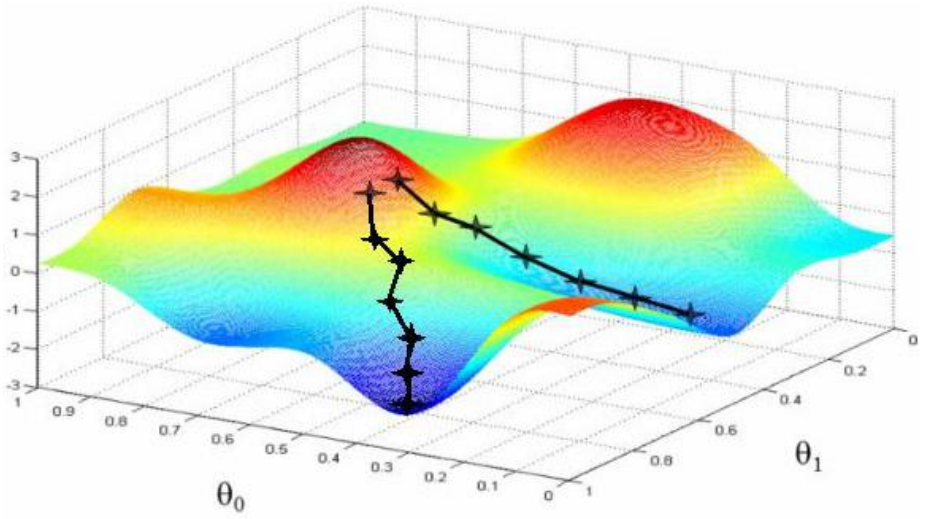


Why is it **Stochastic** Gradient Descent?

- Initialize θ randomly
- For N Epochs
  - For each training example (x, y):
    - Compute Loss Gradient: $\frac{\partial J(\theta)}{\partial \theta}$
    - Update θ with update rule:

$$\theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$$

Only an estimate of true gradient!



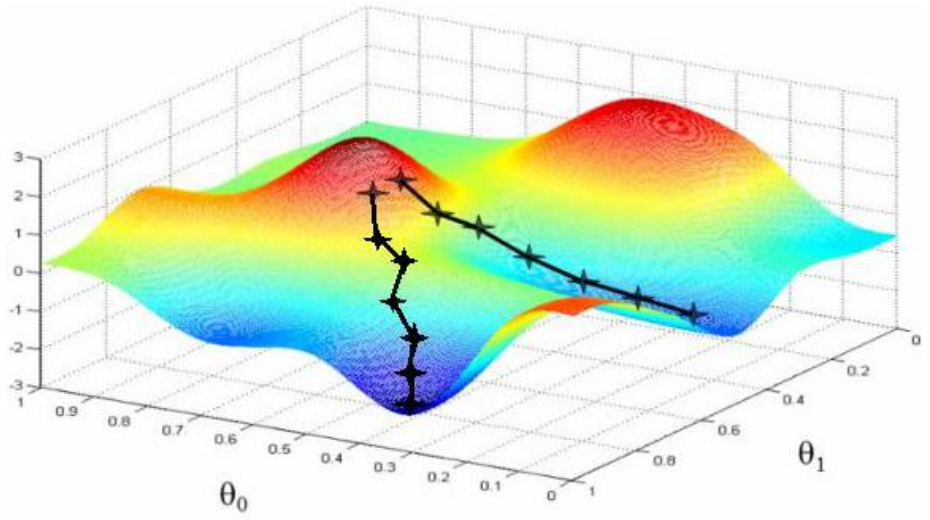
Why is it **Stochastic** Gradient Descent?

- Initialize θ randomly
- For N Epochs
  - For each training batch $\{(x_0, y_0), \ldots, (x_B, y_B)\}$:

More accurate estimate!

- Compute Loss Gradient: $\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{B} \sum_{i}^{B} \frac{\partial J_i(\theta)}{\partial \theta}$
- Update θ with update rule:

$$\theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$$



Advantages:

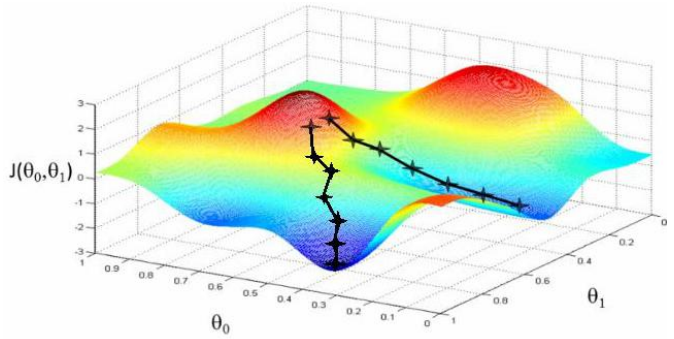
- More accurate estimation of gradient
  - Smoother convergence
  - Allows for larger learning rates
- Minibatches lead to fast training!
  - Can parallelize computation + achieve significant speed increases on GPU's

Stochastic Gradient Descent (SGD)

- Algorithm that performs updates after each example
  - —initialize $\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$
  - —for N iterations
    - —for each training example $(\mathbf{x}^{(t)}, y^{(t)})$ or batch

$$\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \, \Delta$$



Training epoch
=
Iteration over **all** examples

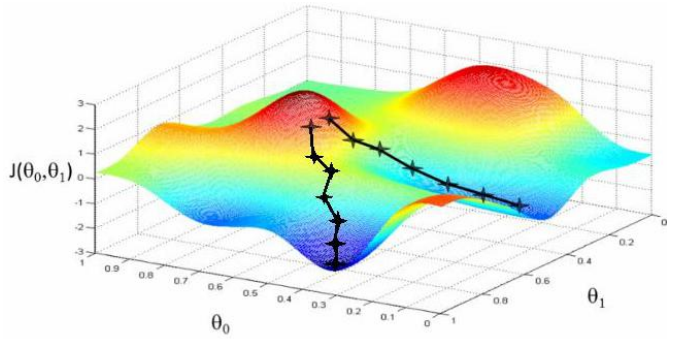
- To apply this algorithm to neural network training, we need:
  - —the loss function $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - —a procedure to compute the parameter gradients: $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - —the regularizer $\Omega(\boldsymbol{\theta})$ (and the gradient $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$)

Stochastic Gradient Descent (SGD)

- Algorithm that performs updates after each example
  - —initialize $\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$
  - —for N iterations
    - —for each training example $(\mathbf{x}^{(t)}, y^{(t)})$ or batch

$$\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \, \Delta$$



Training epoch
=
Iteration over **all** examples

- To apply this algorithm to neural network training, we need:
  - —the loss function $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - —a procedure to compute the parameter gradients: $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - —the regularizer $\Omega(\boldsymbol{\theta})$ (and the gradient $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$)

What is a neural network again?

- A family of parametric, non-linear and hierarchical representation learning functions

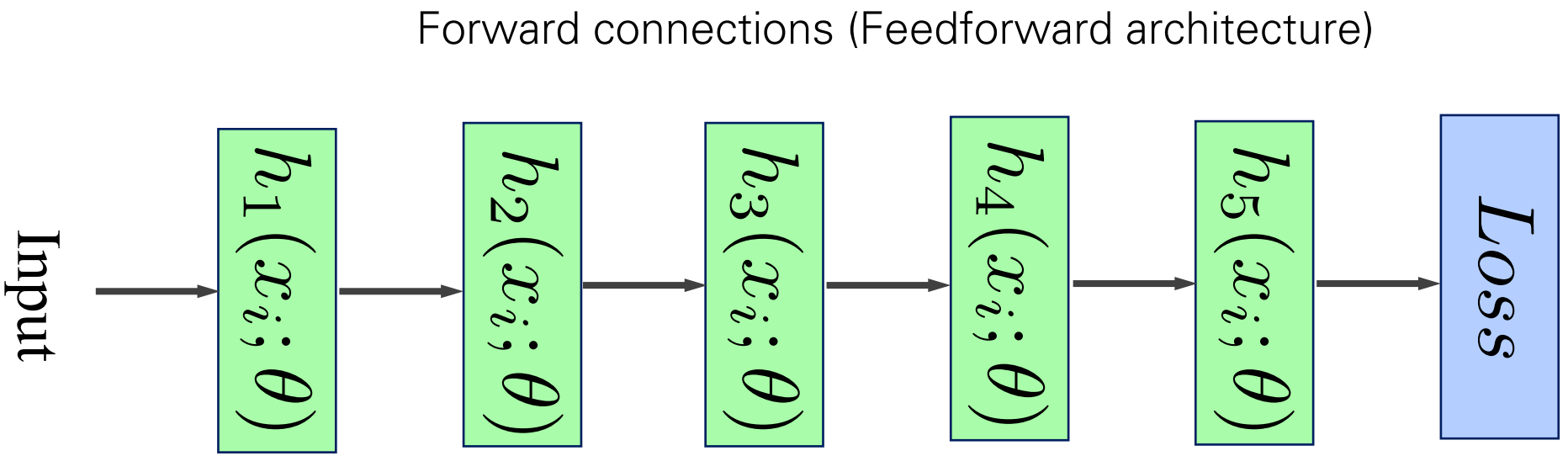
- $a_L(x; \theta_{1,\ldots,L}) = h_L(h_{L-1}(\ldots h_1(x, \theta_1), \theta_{L-1}), \theta_L)$
  - x: input, θ_l: parameters for layer l, $a_l = h_l(x, \theta_l)$: (non-)linear function

- Given training corpus $\{X, Y\}$ find optimal parameters

$$\theta^* \leftarrow \arg\min_{\theta} \sum_{(x,y)\in(X,Y)} \ell(y, a_L(x; \theta_{1,\ldots,L}))$$

Neural network models

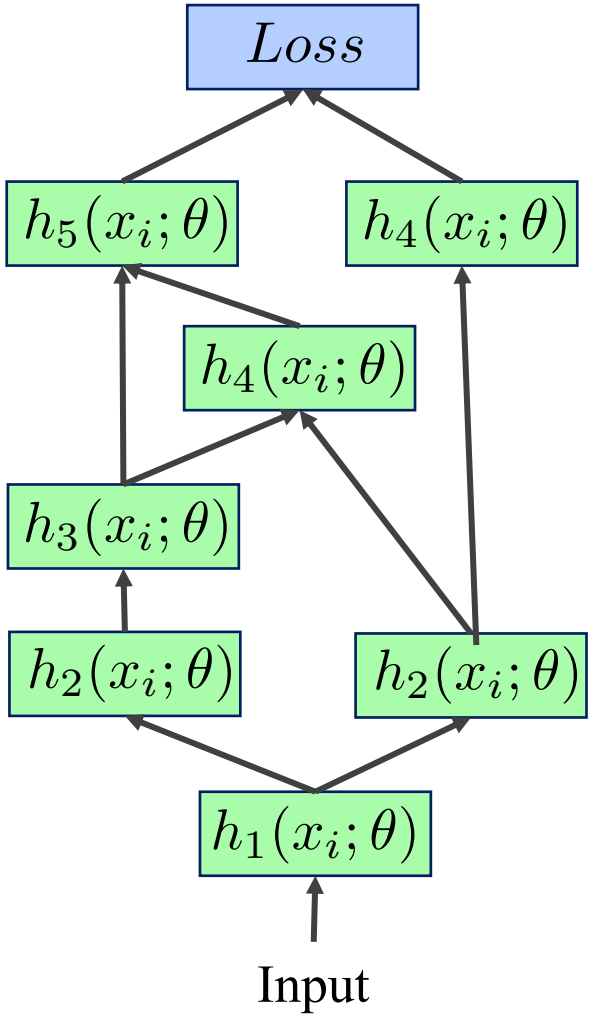
- A neural network model is a series of hierarchically connected functions
- The hierarchy can be very, very complex



Neural network models

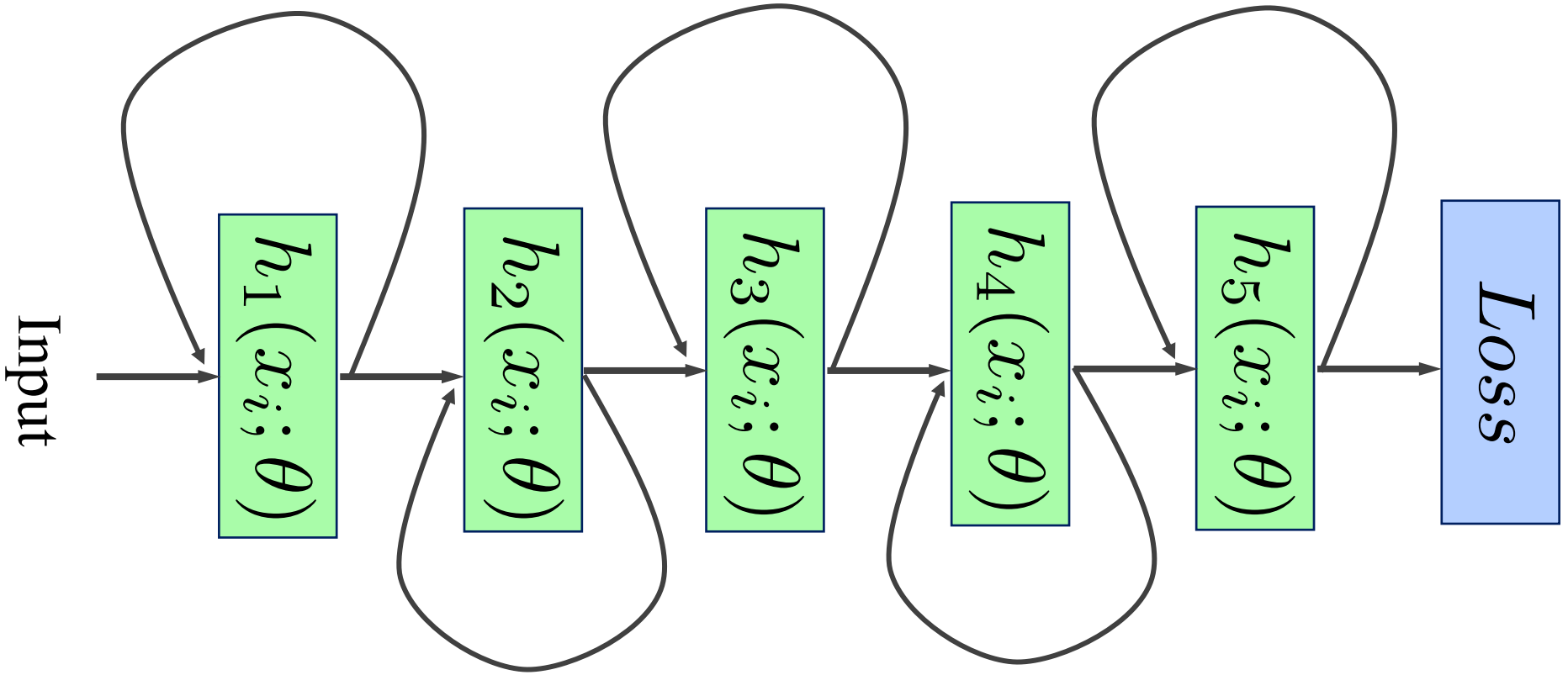
- A neural network model is a series of hierarchically connected functions
- The hierarchy can be very, very complex

Interweaved connections (Directed Acyclic Graph architecture – DAGNN)



Neural network models

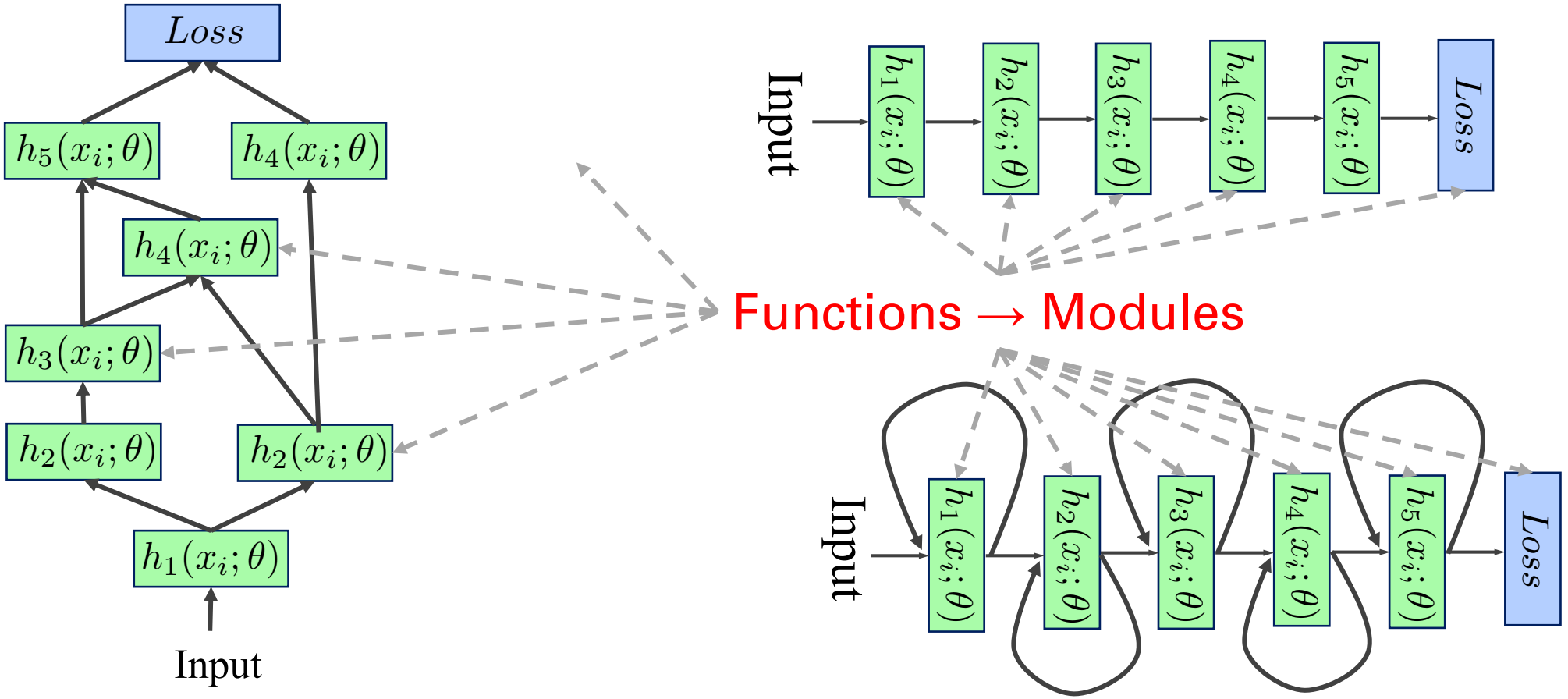
- A neural network model is a series of hierarchically connected functions
- The hierarchy can be very, very complex



Loopy connections (Recurrent architecture, special care needed)

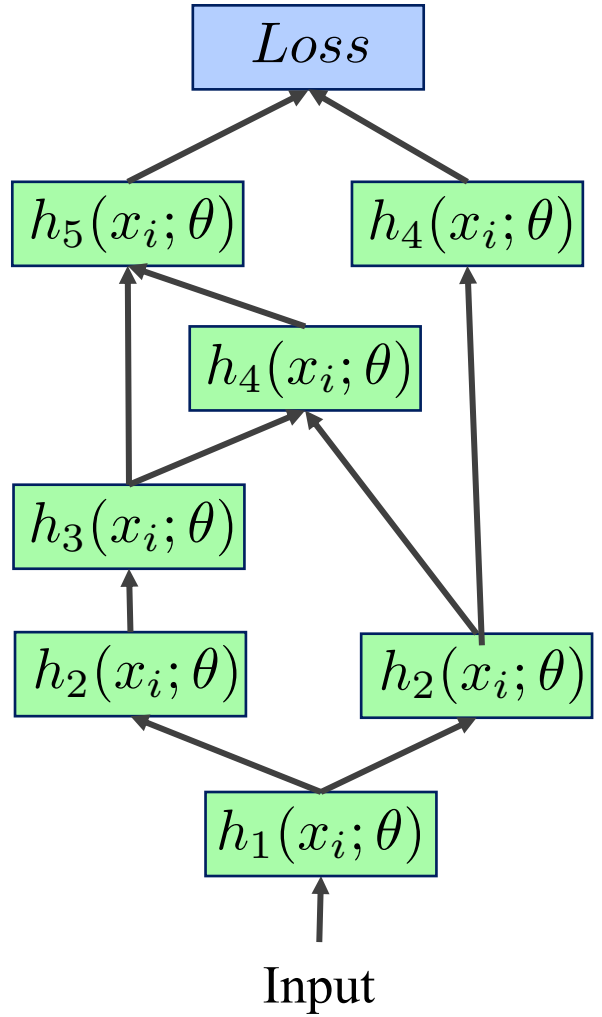
Neural network models

- A neural network model is a series of hierarchically connected functions
- The hierarchy can be very, very complex



What is a module

- A module is a building block for our network
- Each module is an object/function $a = h(x; \theta)$ that
  - Contains trainable parameters (θ)
  - Receives as an argument an input x
  - And returns an output a based on the activation function $h(...)$
- The activation function should be (at least) first order differentiable (almost) everywhere
- For easier/more efficient backpropagation
  → store module input
  - easy to get module output fast
  - easy to compute derivatives



Anything goes or do special constraints exist?

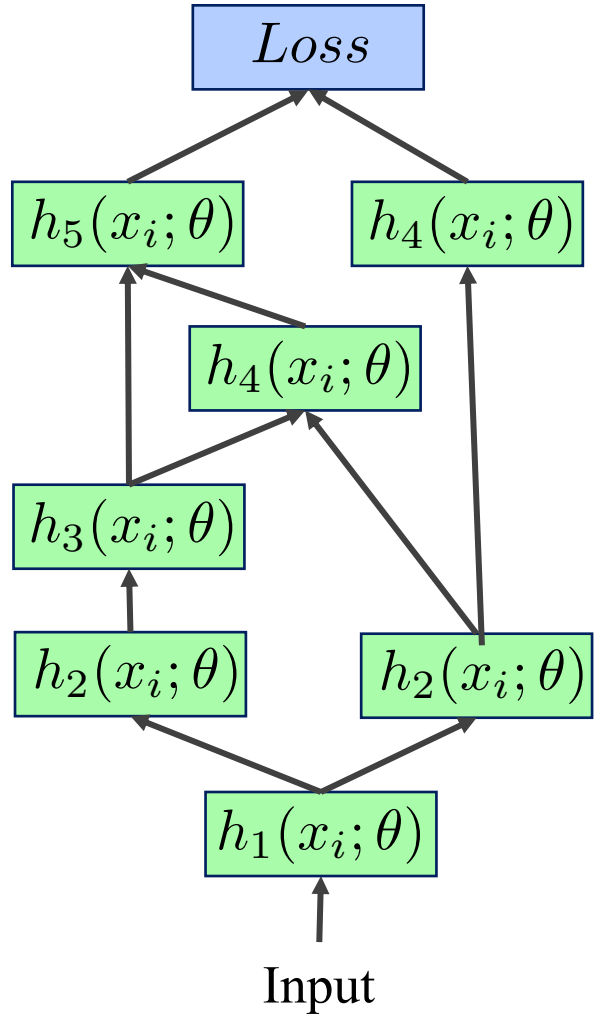
- A neural network is a composition of modules (building blocks)
- Any architecture works
- If the architecture is a feedforward cascade, no special care
- If acyclic, there is right order of computing the forward computations
- If there are loops, these form **recurrent** connections (revisited later)

What is a module

- Simply compute the activation of each module in the network

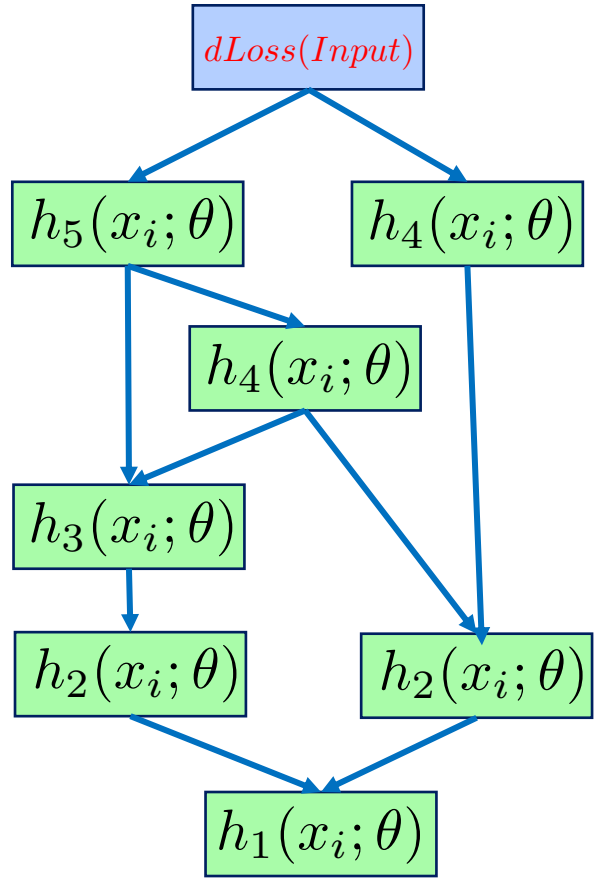
$$a_l = h_l(x_l; \theta) \text{ where } a_l = x_{l+1} \text{ or } x_l = a_{l-1}$$

- We need to know the precise function behind each module $h_l(\dots)$
- Recursive operations
  - One module's output is another's input
- Steps
  - Visit modules one by one starting from the data input
  - Some modules might have several inputs from multiple modules
- Compute modules activations **with the right order**
  - Make sure all the inputs computed at the right time



What is a module

- Simply compute the gradients of each module for our data
  - We need to know the gradient formulation of each module $\partial h_l(x_l;\theta_l)$ w.r.t. their inputs x_l and parameters θ_l
- We need the forward computations first
  - Their result is the sum of losses for our input data
- Then take the reverse network (reverse connections) and traverse it backwards
- Instead of using the activation functions, we use their gradients
- The whole process can be described very neatly and concisely with the backpropagation algorithm



Again, what is a neural network again?

- $a_L(x; \theta_{1,\ldots,L}) = h_L(h_{L-1}(\ldots h_1(x, \theta_1), \theta_{L-1}), \theta_L)$
  - x: input, θ_l: parameters for layer l, $a_l = h_l(x, \theta_l)$: (non-)linear function
- Given training corpus $\{X, Y\}$ find optimal parameters

$$\theta^* \leftarrow \arg\min_{\theta} \sum_{(x,y)\in(X,Y)} \ell(y, a_L(x; \theta_{1,\ldots,L}))$$

- To use any gradient descent based optimization

$$\left(\theta^{t+1} = \theta^t - \eta_t \frac{\partial \mathcal{L}}{\partial \theta^t}\right)$$

we need the gradients

$$\frac{\partial \mathcal{L}}{\partial \theta_l}, l = 1, \ldots, L$$

- How to compute the gradients for such a complicated function enclosing other functions, like $a_L(\ldots)$?

Again, what is a neural network again?

- $a_L(x; \theta_{1,\ldots,L}) = h_L(h_{L-1}(\ldots h_1(x, \theta_1), \theta_{L-1}), \theta_L)$
  - x: input, θ_l: parameters for layer l, $a_l = h_l(x, \theta_l)$: (non-)linear function
- Given training corpus $\{X, Y\}$ find optimal parameters

$$\theta^* \leftarrow \arg\min_{\theta} \sum_{(x,y)\in(X,Y)} \ell(y, a_L(x; \theta_{1,\ldots,L}))$$

- To use any gradient descent based optimization

$$\left(\theta^{t+1} = \theta^t - \eta_t \frac{\partial \mathcal{L}}{\partial \theta^t}\right)$$

we need the gradients

$$\frac{\partial \mathcal{L}}{\partial \theta_l}, l = 1, \ldots, L$$

- How to compute the gradients for such a complicated function enclosing other functions, like a_L (…)?

How do we compute gradients?

- Numerical Differentiation
- Symbolic Differentiation
- Automatic Differentiation (AutoDiff)

Numerical Differentiation

$\mathbf{1}_i$ - Vector of all zeros, except for one 1 in i-th location

- We can approximate the gradient numerically, using:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \to 0} \frac{f(\mathbf{x} + h\mathbf{1}_i) - f(\mathbf{x})}{h}$$

Numerical Differentiation

$\mathbf{1}_i$ - Vector of all zeros, except for one 1 in i-th location

- We can approximate the gradient numerically, using:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \to 0} \frac{f(\mathbf{x} + h\mathbf{1}_i) - f(\mathbf{x})}{h}$$

- Even better, we can use central differencing:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \to 0} \frac{f(\mathbf{x} + h\mathbf{1}_i) - f(\mathbf{x} - h\mathbf{1}_i)}{2h}$$

Numerical Differentiation

$\mathbf{1}_i$ - Vector of all zeros, except for one 1 in i-th location

- We can approximate the gradient numerically, using:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \to 0} \frac{f(\mathbf{x} + h\mathbf{1}_i) - f(\mathbf{x})}{h}$$

- Even better, we can use central differencing:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \to 0} \frac{f(\mathbf{x} + h\mathbf{1}_i) - f(\mathbf{x} - h\mathbf{1}_i)}{2h}$$

- However, both of these suffer from rounding errors and are not good enough for learning (they are very good tools for checking the correctness of implementation though, e.g., use $h = 0.000001$).

Numerical Differentiation

$\mathbf{1}_i$ - Vector of all zeros, except for one 1 in i-th location

$\mathbf{1}_{ij}$ - Matrix of all zeros, except for one 1 in (i,j)-th location

- We can approximate the gradient numerically, using:

$$\frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial w_{ij}} \approx \lim_{h \to 0} \frac{\mathcal{L}(\mathbf{W} + h\mathbf{1}_{ij}, \mathbf{b}) - \mathcal{L}(\mathbf{W}, \mathbf{b})}{h} \qquad \frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial b_j} \approx \lim_{h \to 0} \frac{\mathcal{L}(\mathbf{W}, \mathbf{b} + h\mathbf{1}_j) - \mathcal{L}(\mathbf{W}, \mathbf{b})}{h}$$

- Even better, we can use central differencing:

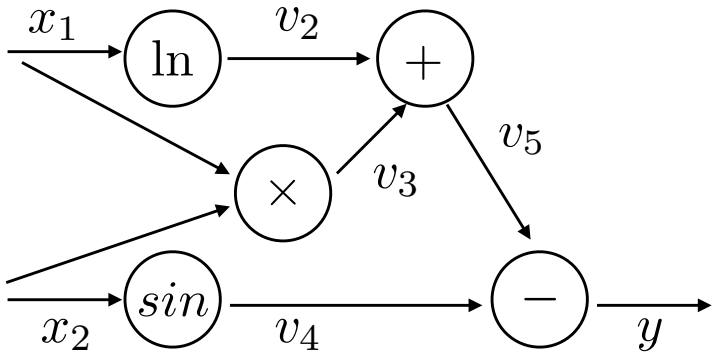
$$\frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial w_{ij}} \approx \lim_{h \to 0} \frac{\mathcal{L}(\mathbf{W} + h\mathbf{1}_{ij}, \mathbf{b}) - \mathcal{L}(\mathbf{W} + h\mathbf{1}_{ij}, \mathbf{b})}{2h} \qquad \frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial b_j} \approx \lim_{h \to 0} \frac{\mathcal{L}(\mathbf{W}, \mathbf{b} + h\mathbf{1}_j) - \mathcal{L}(\mathbf{W}, \mathbf{b} + h\mathbf{1}_j)}{2h}$$

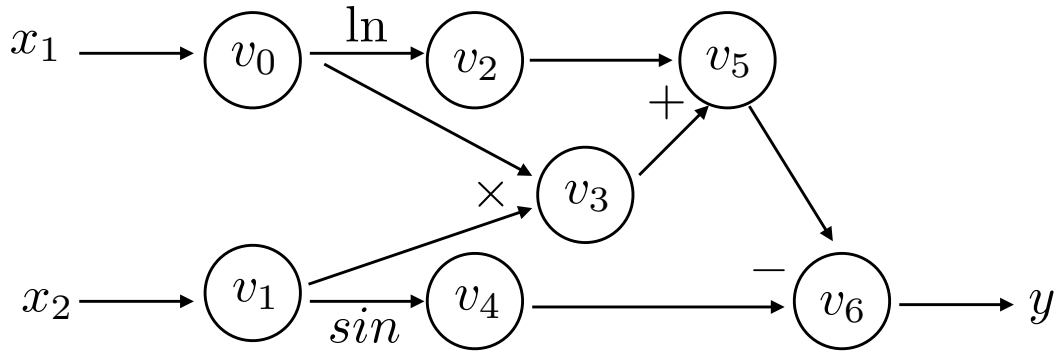
- However, both of these suffer from rounding errors and are not good enough for learning (they are very good tools for checking the correctness of implementation though, e.g., use $h = 0.000001$).

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

Symbolic Differentiation

- Input function is represented as **computational graph** (a symbolic tree)





- Implements differentiation rules for composite functions:

Sum Rule

$$\frac{\mathrm{d}\left(f(x)+g(x)\right)}{\mathrm{d}x} = \frac{\mathrm{d}f(x)}{\mathrm{d}x} + \frac{\mathrm{d}g(x)}{\mathrm{d}x}$$

Product Rule

$$\frac{\mathrm{d}\left(f(x)\cdot g(x)\right)}{\mathrm{d}x} = \frac{\mathrm{d}f(x)}{\mathrm{d}x} g(x) + f(x)\frac{\mathrm{d}g(x)}{\mathrm{d}x}$$

Chain Rule

$$\frac{\mathrm{d}(f(g(x)))}{\mathrm{d}x} = \frac{\mathrm{d}f(g(x))}{\mathrm{d}x} \cdot \frac{\mathrm{d}g(x)}{\mathrm{d}x}$$

Problem: For complex functions, expressions can be exponentially large; also difficult to deal with piece-wise functions (creates many symbolic cases)

Automatic Differentiation (AutoDiff)

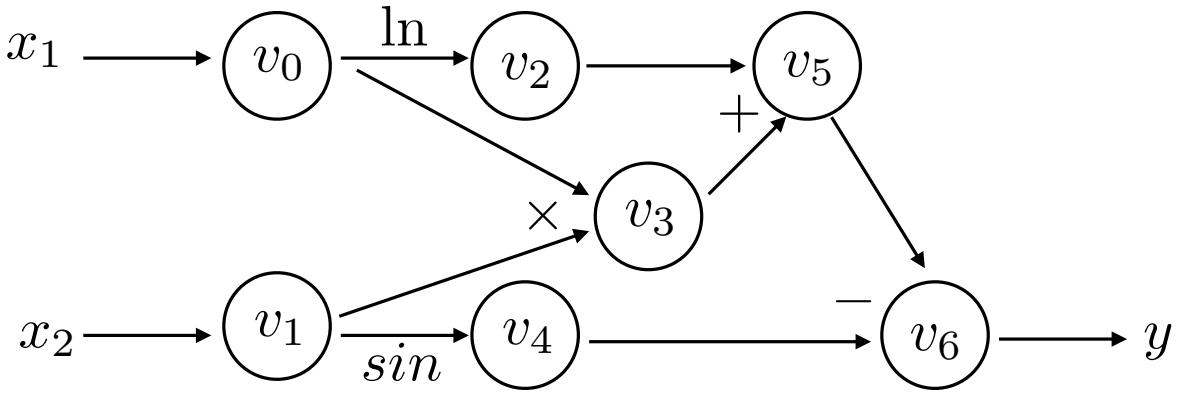
- **Intuition:** Interleave symbolic differentiation and simplification
- **Key Idea:** Apply symbolic differentiation at the elementary operation level, evaluate and keep intermediate results

Success of **deep learning** owes A LOT to success of AutoDiff algorithms
(also to advances in parallel architectures, and large datasets, ...)

slide adopted from T. Chen, H. Shen, A. Krishnamurthy

Automatic Differentiation (AutoDiff)

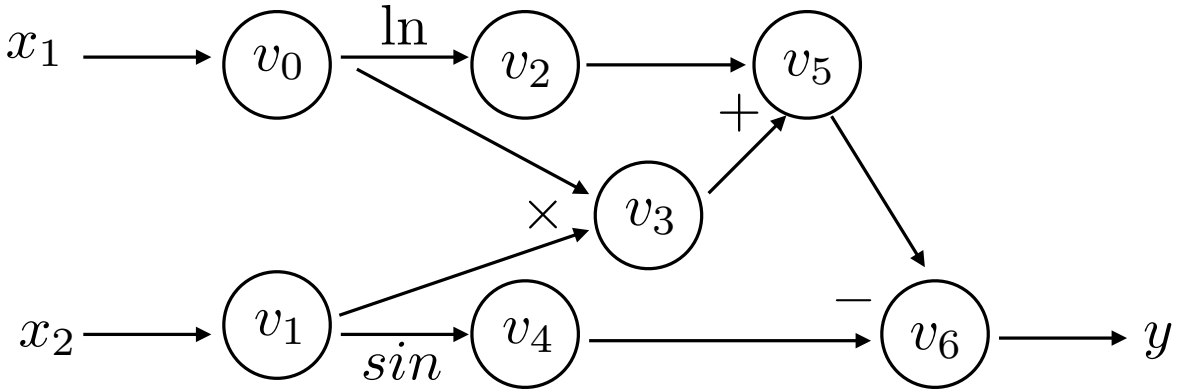
$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



- Each **node** is an input, intermediate, or output variable
- **Computational graph** (a DAG) with variable ordering from topological sort.

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



- Each **node** is an input, intermediate, or output variable
- **Computational graph** (a DAG) with variable ordering from topological sort.

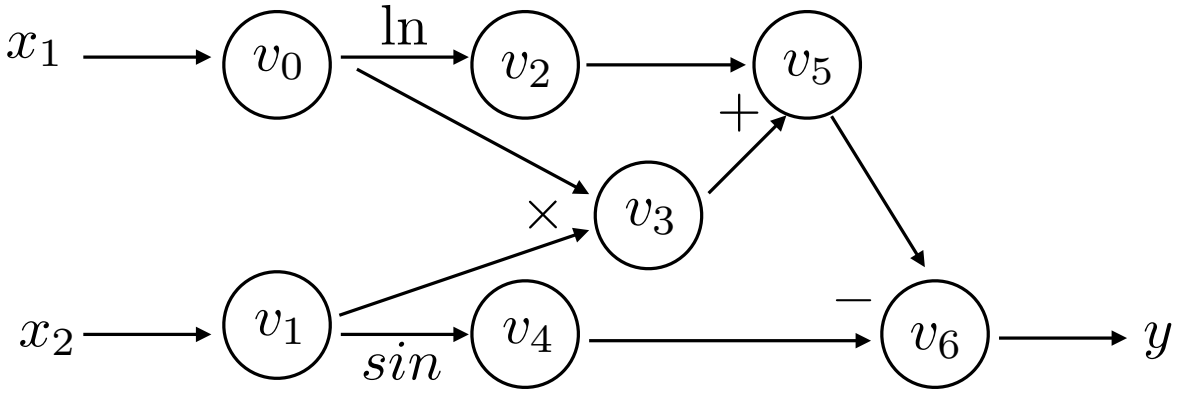
Computational graph is governed by these equations:

$$v_0 = x_1$$
$$v_1 = x_2$$
$$v_2 = \ln(v_0)$$
$$v_3 = v_0 \cdot v_1$$
$$v_4 = sin(v_1)$$
$$v_5 = v_2 + v_3$$
$$v_6 = v_5 - v_4$$
$$y = v_6$$

slide adopted from T. Chen, H. Shen, A. Krishnamurthy

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



- Each **node** is an input, intermediate, or output variable
- **Computational graph** (a DAG) with variable ordering from topological sort.

Let's see how we can **evaluate a function** using computational graph (DNN inferences)

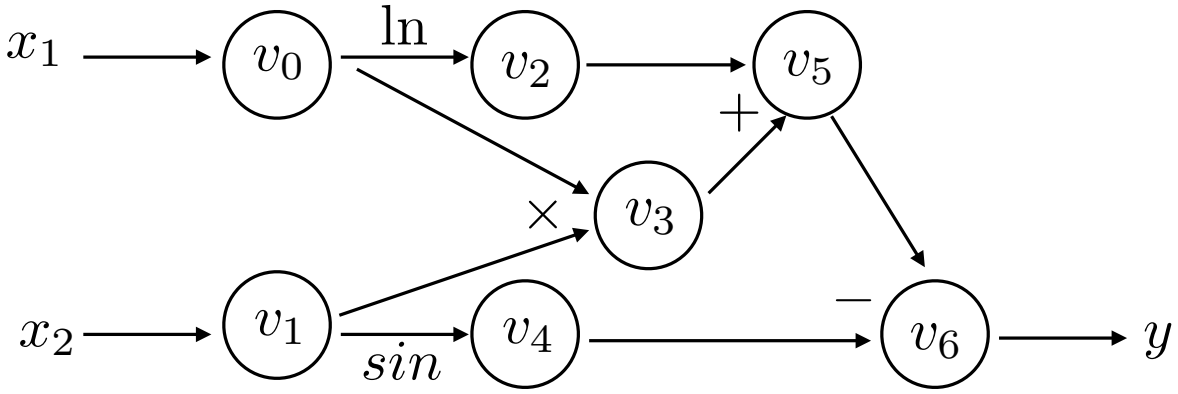
Computational graph is governed by these equations:

$$v_0 = x_1$$
$$v_1 = x_2$$
$$v_2 = \ln(v_0)$$
$$v_3 = v_0 \cdot v_1$$
$$v_4 = sin(v_1)$$
$$v_5 = v_2 + v_3$$
$$v_6 = v_5 - v_4$$
$$y = v_6$$

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

Let's see how we can **evaluate a function** using computational graph (DNN inferences)



- Each **node** is an input, intermediate, or output variable
- **Computational graph** (a DAG) with variable ordering from topological sort.

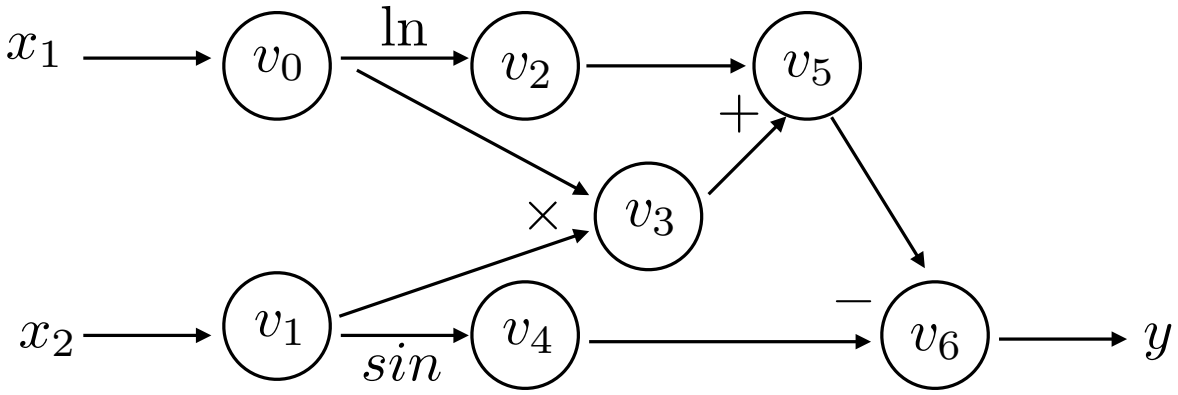
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	
$v_1 = x_2$	
$v_2 = \ln(v_0)$	
$v_3 = v_0 \cdot v_1$	
$v_4 = sin(v_1)$	
$v_5 = v_2 + v_3$	
$v_6 = v_5 - v_4$	
$y = v_6$	

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

Let's see how we can **evaluate a function** using computational graph (DNN inferences)



- Each **node** is an input, intermediate, or output variable
- **Computational graph** (a DAG) with variable ordering from topological sort.

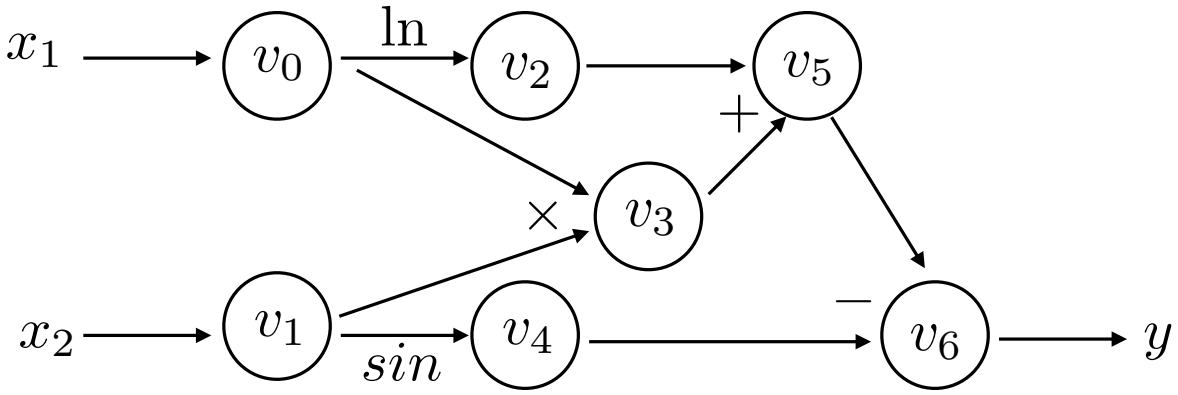
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	
$v_2 = \ln(v_0)$	
$v_3 = v_0 \cdot v_1$	
$v_4 = sin(v_1)$	
$v_5 = v_2 + v_3$	
$v_6 = v_5 - v_4$	
$y = v_6$	

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

Let's see how we can **evaluate a function** using computational graph (DNN inferences)



- Each **node** is an input, intermediate, or output variable
- **Computational graph** (a DAG) with variable ordering from topological sort.

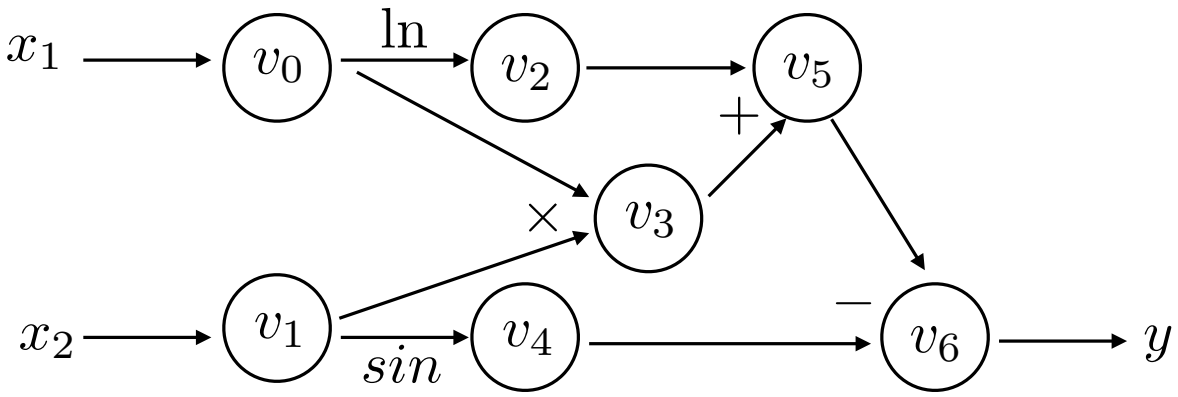
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	
$v_3 = v_0 \cdot v_1$	
$v_4 = sin(v_1)$	
$v_5 = v_2 + v_3$	
$v_6 = v_5 - v_4$	
$y = v_6$	

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

Let's see how we can **evaluate a function** using computational graph (DNN inferences)



- Each **node** is an input, intermediate, or output variable
- **Computational graph** (a DAG) with variable ordering from topological sort.

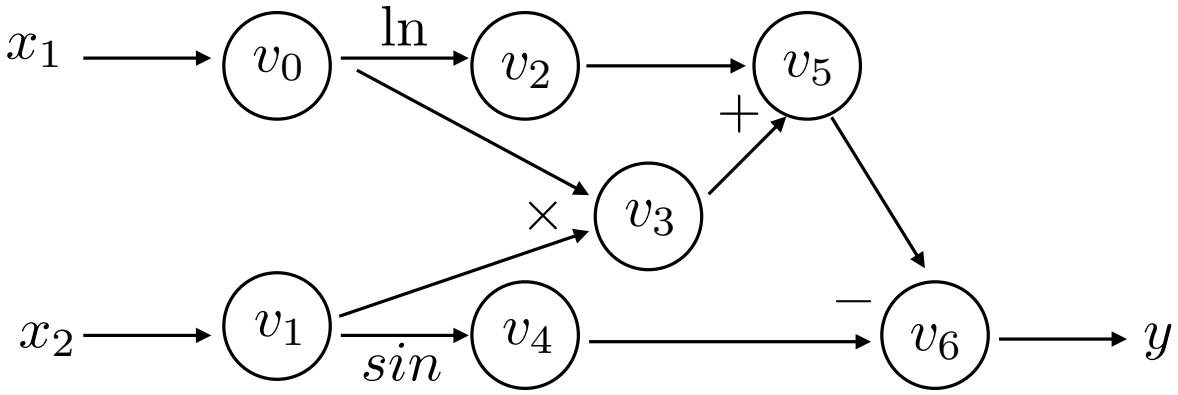
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	
$v_4 = sin(v_1)$	
$v_5 = v_2 + v_3$	
$v_6 = v_5 - v_4$	
$y = v_6$	

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

Let's see how we can **evaluate a function** using computational graph (DNN inferences)



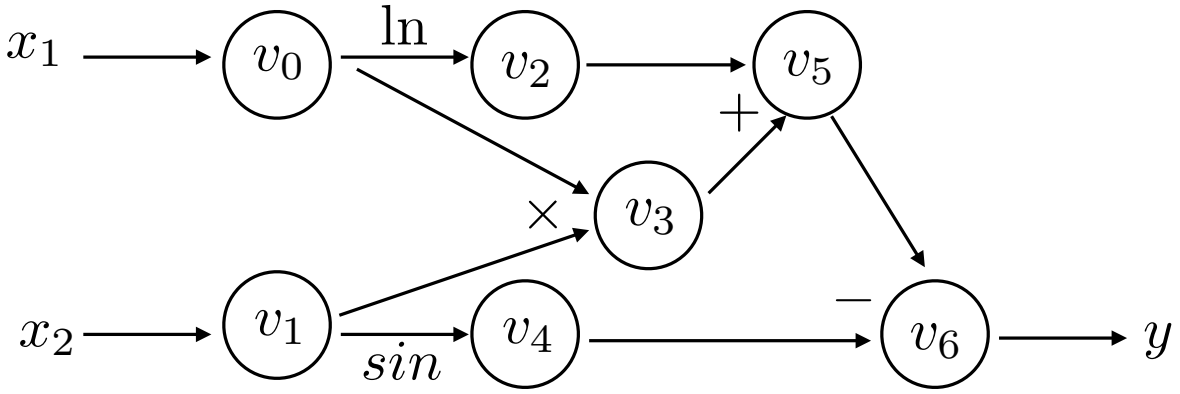
- Each **node** is an input, intermediate, or output variable
- **Computational graph** (a DAG) with variable ordering from topological sort.

Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	
$v_5 = v_2 + v_3$	
$v_6 = v_5 - v_4$	
$y = v_6$	

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



- Each **node** is an input, intermediate, or output variable
- **Computational graph** (a DAG) with variable ordering from topological sort.

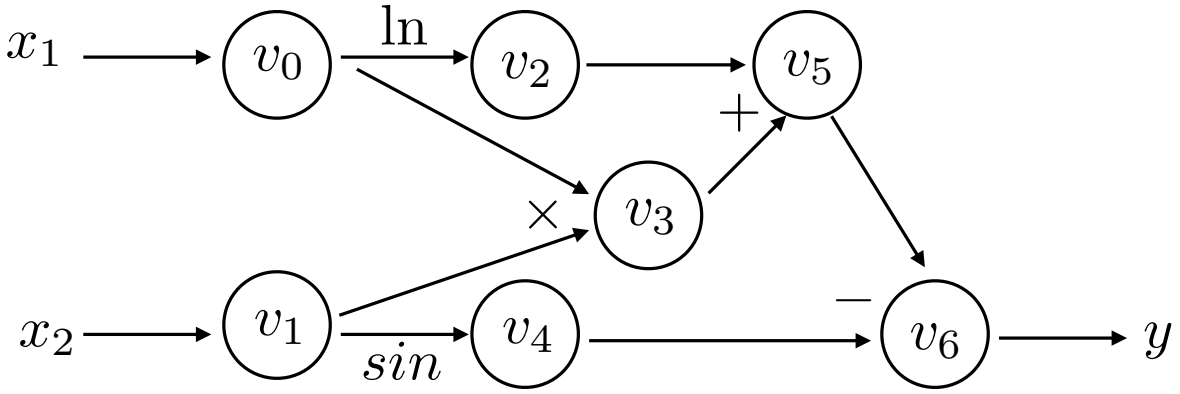
Let's see how we can **evaluate a function** using computational graph (DNN inferences)

Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	
$v_6 = v_5 - v_4$	
$y = v_6$	

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



- Each **node** is an input, intermediate, or output variable
- **Computational graph** (a DAG) with variable ordering from topological sort.

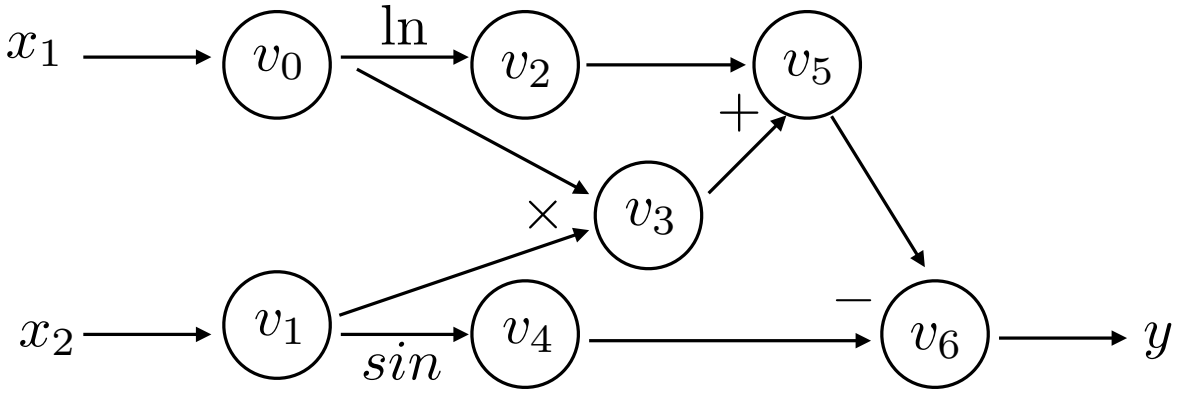
Let's see how we can **evaluate a function** using computational graph (DNN inferences)

Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	
$y = v_6$	

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



- Each **node** is an input, intermediate, or output variable
- **Computational graph** (a DAG) with variable ordering from topological sort.

Let's see how we can **evaluate a function** using computational graph (DNN inferences)

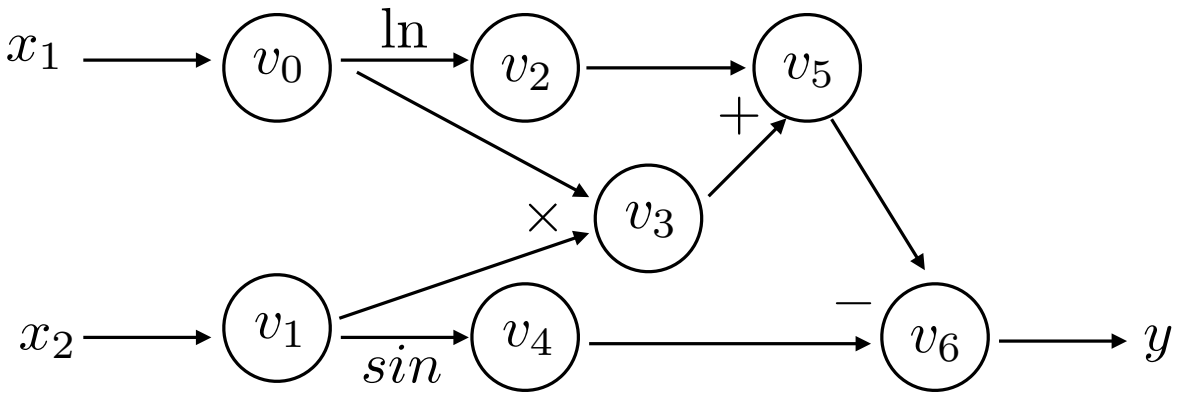
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

Let's see how we can **evaluate a function** using computational graph (DNN inferences)



- Each **node** is an input, intermediate, or output variable
- **Computational graph** (a DAG) with variable ordering from topological sort.

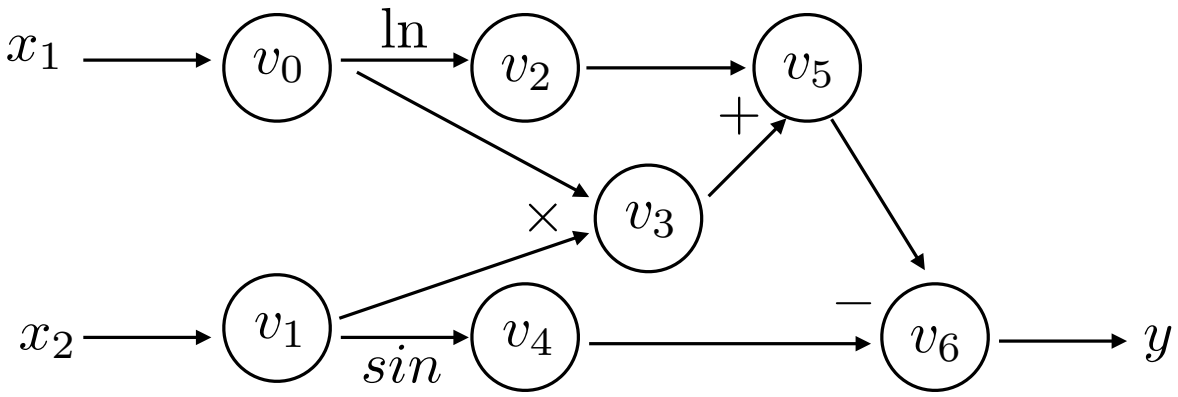
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

Let's see how we can **evaluate a function** using computational graph (DNN inferences)



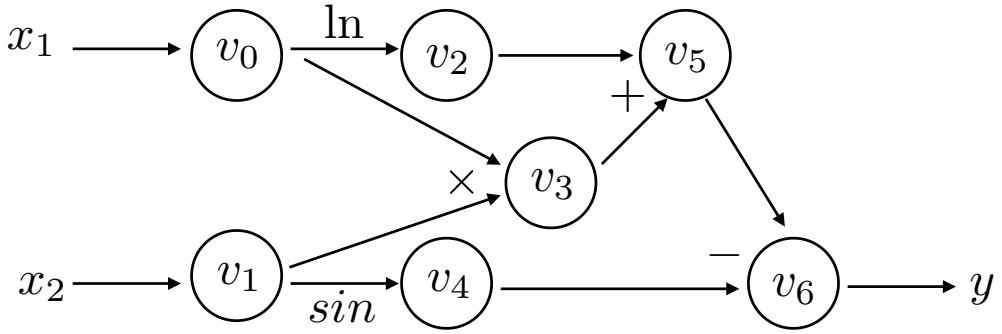
- Each **node** is an input, intermediate, or output variable
- **Computational graph** (a DAG) with variable ordering from topological sort.

Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Automatic Differentiation (AutoDiff)

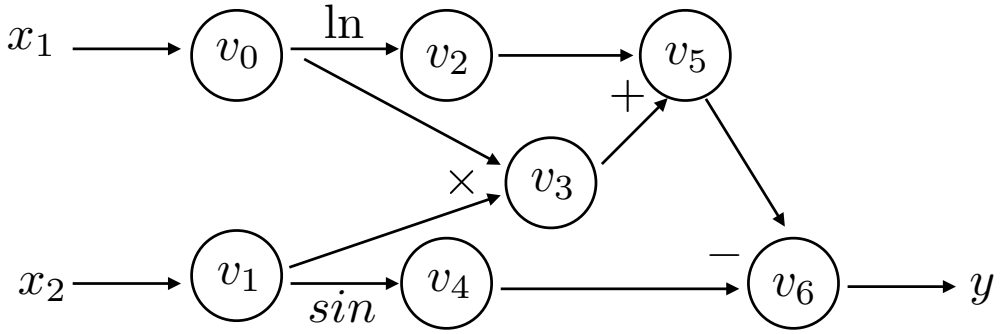
$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Automatic Differentiation (AutoDiff)



$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

Let's see how we can **evaluate a function** using computational graph (DNN inferences)

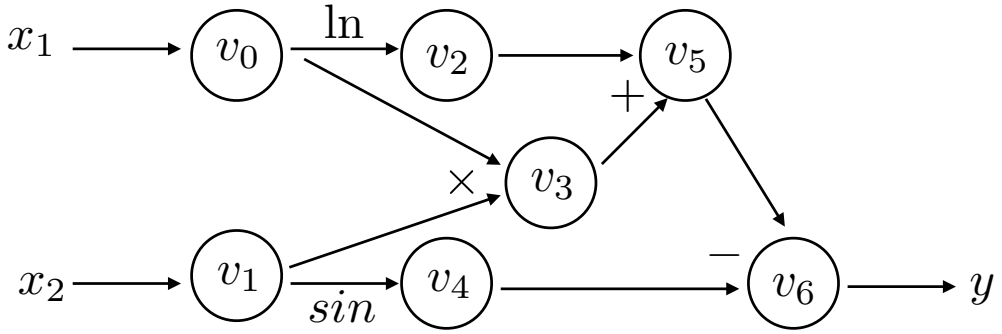
$$\left.\frac{\partial f(x_1, x_2)}{\partial x_1}\right|_{(x_1=2, x_2=5)}$$

We will do this with **forward mode** first, by introducing a derivative of each variable node with respect to the input variable.

Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Automatic Differentiation (AutoDiff)



$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

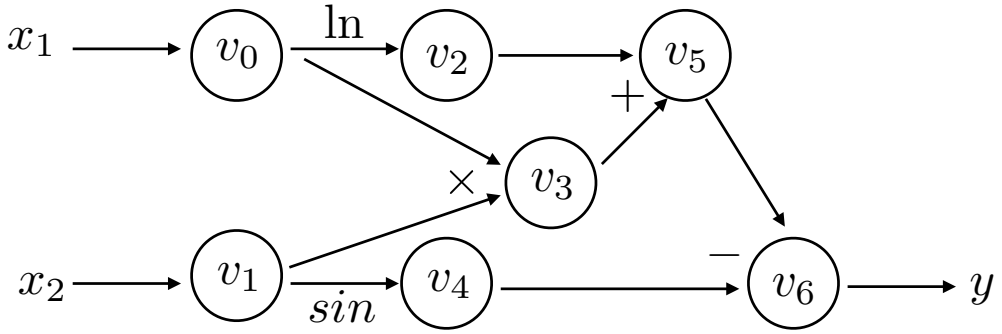
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\frac{\partial f(x_1, x_2)}{\partial x_1}\Big\|_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	

Automatic Differentiation (AutoDiff)



$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

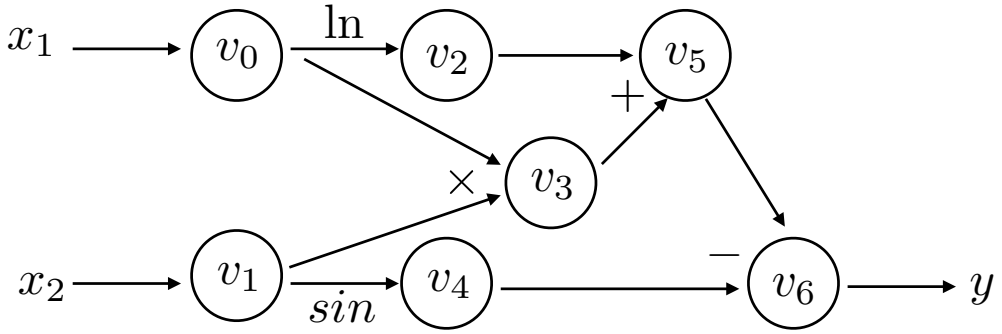
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\frac{\partial f(x_1, x_2)}{\partial x_1}\Big\|_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1

Automatic Differentiation (AutoDiff)



$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

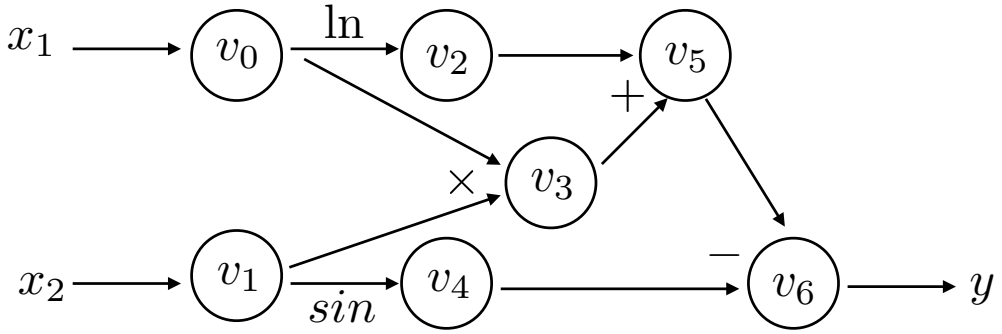
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\frac{\partial f(x_1, x_2)}{\partial x_1}\Big\|_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	

Automatic Differentiation (AutoDiff)



$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

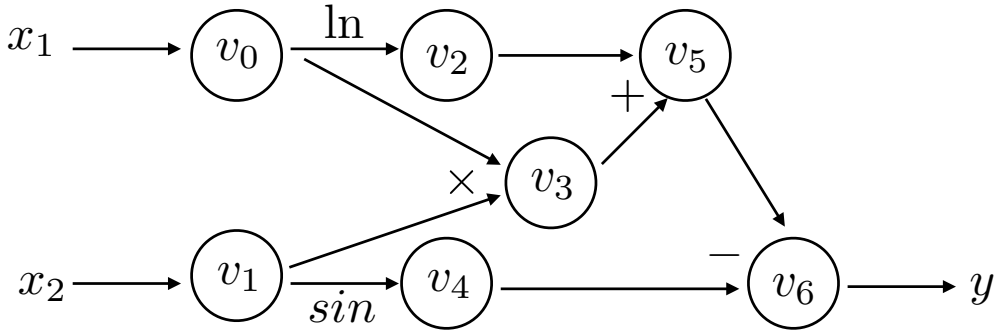
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\frac{\partial f(x_1, x_2)}{\partial x_1}\Big\|_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0

Automatic Differentiation (AutoDiff)



$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

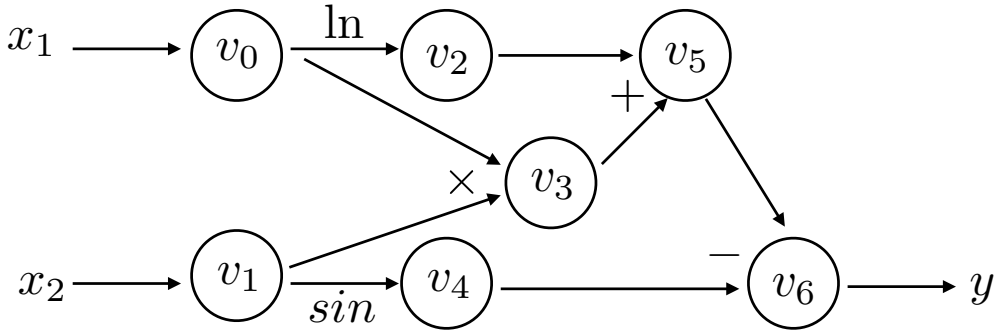
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\frac{\partial f(x_1, x_2)}{\partial x_1}\Big\|_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1}$	

Automatic Differentiation (AutoDiff)



$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

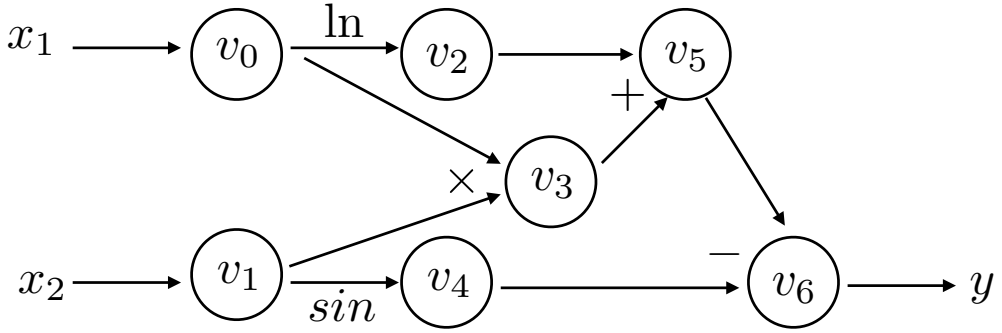
Forward Derivative Trace:

	$\frac{\partial f(x_1, x_2)}{\partial x_1}\Big\|_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1}$	

Chain Rule

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



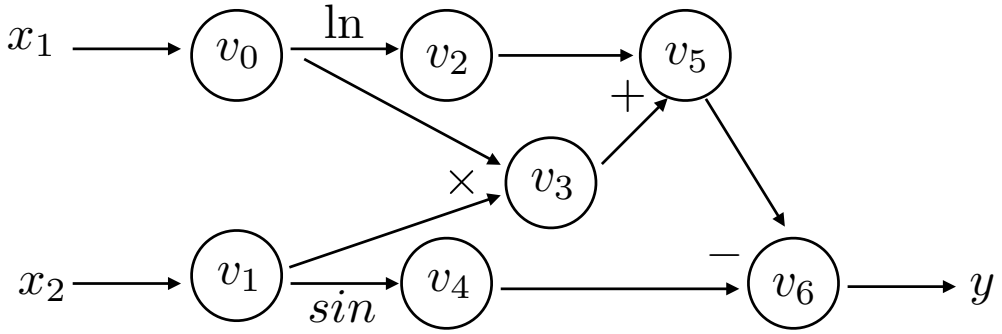
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\frac{\partial f(x_1, x_2)}{\partial x_1}\Big\|_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1} = \frac{1}{v_0}\frac{\partial v_0}{\partial x_1}$ Chain Rule	

Automatic Differentiation (AutoDiff)



$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

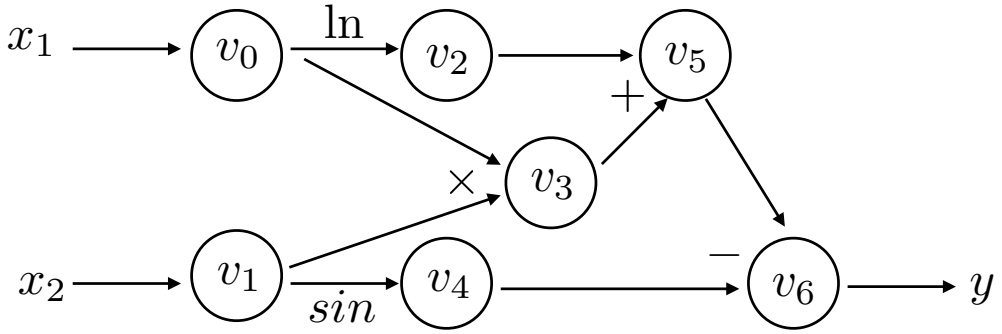
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\left.\frac{\partial f(x_1, x_2)}{\partial x_1}\right\|_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1} = \frac{1}{v_0}\frac{\partial v_0}{\partial x_1}$ Chain Rule	1/2 * 1 = 0.5

Automatic Differentiation (AutoDiff)



$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

Forward Evaluation Trace:

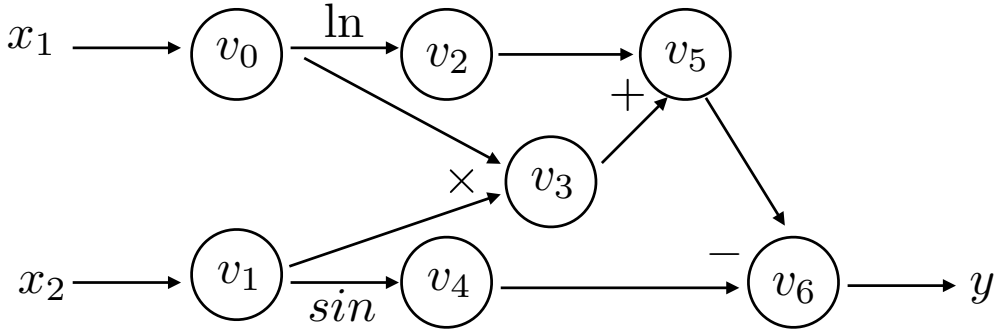
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\left.\frac{\partial f(x_1, x_2)}{\partial x_1}\right\|_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1} = \frac{1}{v_0}\frac{\partial v_0}{\partial x_1}$	1/2 * 1 = 0.5
$\frac{\partial v_3}{\partial x_1}$	

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

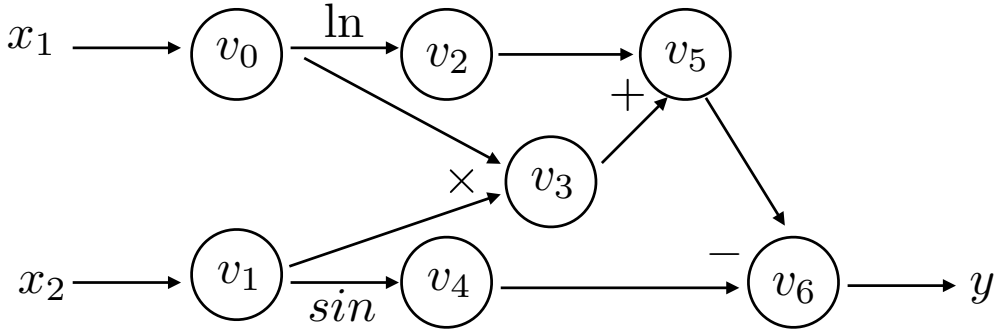
Forward Derivative Trace:

	$\frac{\partial f(x_1, x_2)}{\partial x_1}\Big\|_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1} = \frac{1}{v_0}\frac{\partial v_0}{\partial x_1}$	1/2 * 1 = 0.5
$\frac{\partial v_3}{\partial x_1}$	

Product Rule

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

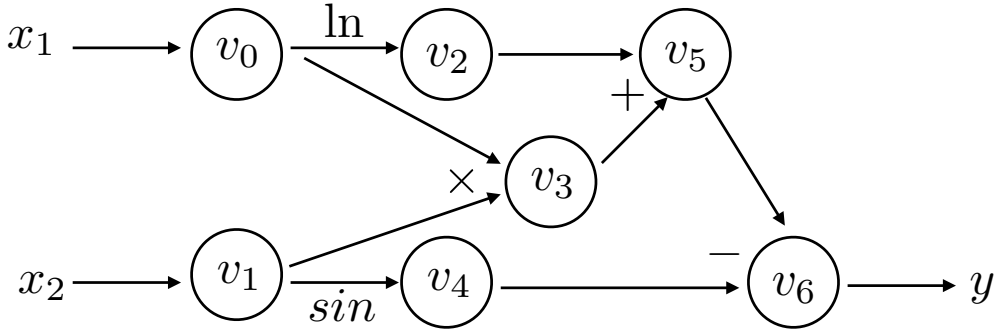
Forward Derivative Trace:

	$\left.\frac{\partial f(x_1, x_2)}{\partial x_1}\right\|_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1} = \frac{1}{v_0}\frac{\partial v_0}{\partial x_1}$	1/2 * 1 = 0.5
$\frac{\partial v_3}{\partial x_1} = \frac{\partial v_0}{\partial x_1} \cdot v_1 + v_0 \cdot \frac{\partial v_1}{\partial x_1}$	

Product Rule

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Forward Evaluation Trace:

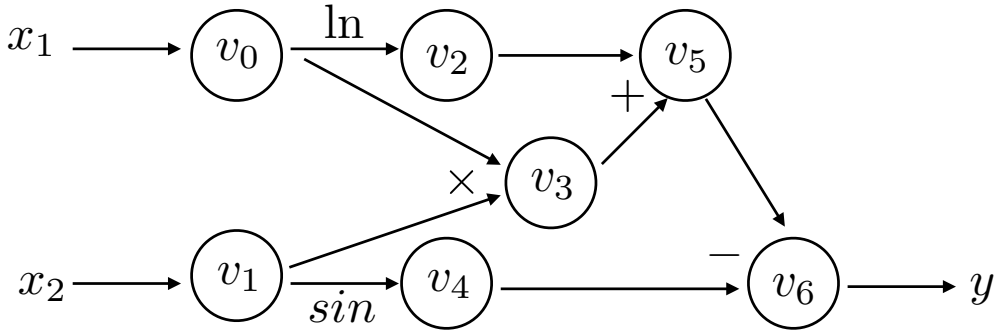
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\frac{\partial f(x_1, x_2)}{\partial x_1}\Big\|_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1} = \frac{1}{v_0}\frac{\partial v_0}{\partial x_1}$	1/2 * 1 = 0.5
$\frac{\partial v_3}{\partial x_1} = \frac{\partial v_0}{\partial x_1} \cdot v_1 + v_0 \cdot \frac{\partial v_1}{\partial x_1}$	1*5 + 2*0 = 5

Product Rule

Automatic Differentiation (AutoDiff)



$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

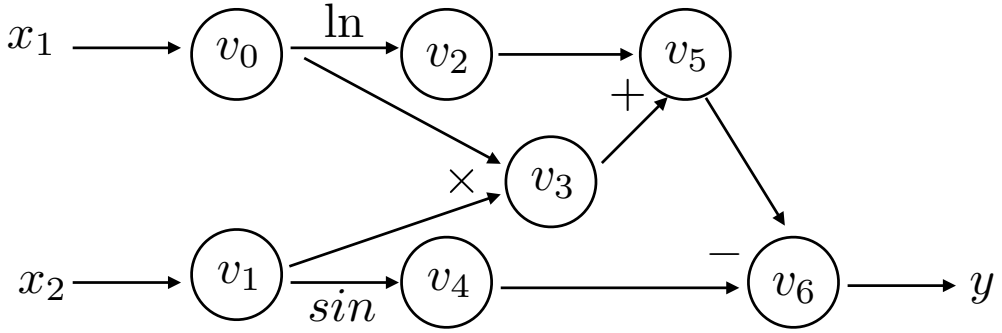
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\left.\frac{\partial f(x_1, x_2)}{\partial x_1}\right\|_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1} = \frac{1}{v_0}\frac{\partial v_0}{\partial x_1}$	1/2 * 1 = 0.5
$\frac{\partial v_3}{\partial x_1} = \frac{\partial v_0}{\partial x_1} \cdot v_1 + v_0 \cdot \frac{\partial v_1}{\partial x_1}$	1*5 + 2*0 = 5
$\frac{\partial v_4}{\partial x_1} = \frac{\partial v_1}{\partial x_1} cos(v_1)$	0 * cos(5) = 0
$\frac{\partial v_5}{\partial x_1} = \frac{\partial v_2}{\partial x_1} + \frac{\partial v_3}{\partial x_1}$	0.5 + 5 = 5.5
$\frac{\partial v_6}{\partial x_1} = \frac{\partial v_5}{\partial x_1} - \frac{\partial v_4}{\partial x_1}$	5.5 – 0 = 5.5
$\frac{\partial y}{\partial x_1} = \frac{\partial v_6}{\partial x_1}$	5.5

Automatic Differentiation (AutoDiff)



$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

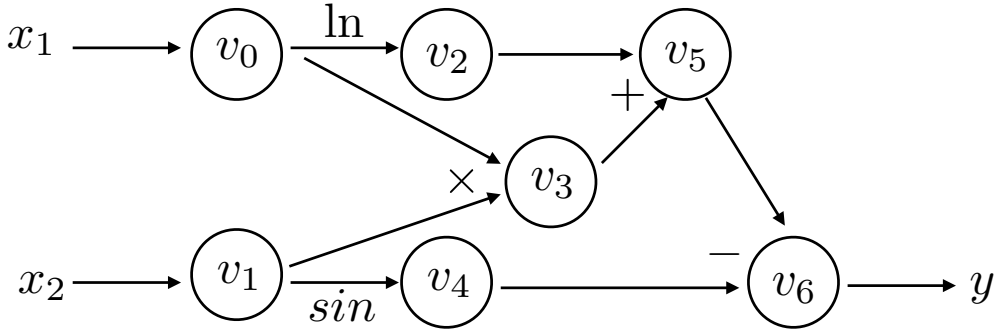
Forward Derivative Trace:

	$\left.\frac{\partial f(x_1, x_2)}{\partial x_1}\right\|_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1} = \frac{1}{v_0}\frac{\partial v_0}{\partial x_1}$	1/2 * 1 = 0.5
$\frac{\partial v_3}{\partial x_1} = \frac{\partial v_0}{\partial x_1} \cdot v_1 + v_0 \cdot \frac{\partial v_1}{\partial x_1}$	1*5 + 2*0 = 5
$\frac{\partial v_4}{\partial x_1} = \frac{\partial v_1}{\partial x_1} cos(v_1)$	0 * cos(5) = 0
$\frac{\partial v_5}{\partial x_1} = \frac{\partial v_2}{\partial x_1} + \frac{\partial v_3}{\partial x_1}$	0.5 + 5 = 5.5
$\frac{\partial v_6}{\partial x_1} = \frac{\partial v_5}{\partial x_1} - \frac{\partial v_4}{\partial x_1}$	5.5 – 0 = 5.5
$\frac{\partial y}{\partial x_1} = \frac{\partial v_6}{\partial x_1}$	5.5

We now have:

$$\left.\frac{\partial f(x_1, x_2)}{\partial x_1}\right|_{(x_1=2, x_2=5)} = 5.5$$

Automatic Differentiation (AutoDiff)



$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

Forward Derivative Trace:

	$\left.\frac{\partial f(x_1, x_2)}{\partial x_1}\right\|_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1} = \frac{1}{v_0}\frac{\partial v_0}{\partial x_1}$	1/2 * 1 = 0.5
$\frac{\partial v_3}{\partial x_1} = \frac{\partial v_0}{\partial x_1} \cdot v_1 + v_0 \cdot \frac{\partial v_1}{\partial x_1}$	1*5 + 2*0 = 5
$\frac{\partial v_4}{\partial x_1} = \frac{\partial v_1}{\partial x_1} cos(v_1)$	0 * cos(5) = 0
$\frac{\partial v_5}{\partial x_1} = \frac{\partial v_2}{\partial x_1} + \frac{\partial v_3}{\partial x_1}$	0.5 + 5 = 5.5
$\frac{\partial v_6}{\partial x_1} = \frac{\partial v_5}{\partial x_1} - \frac{\partial v_4}{\partial x_1}$	5.5 – 0 = 5.5
$\frac{\partial y}{\partial x_1} = \frac{\partial v_6}{\partial x_1}$	5.5

We now have:

$$\left.\frac{\partial f(x_1, x_2)}{\partial x_1}\right|_{(x_1=2, x_2=5)} = 5.5$$

Still need:

$$\left.\frac{\partial f(x_1, x_2)}{\partial x_2}\right|_{(x_1=2, x_2=5)}$$

AutoDiff: Forward Mode

- **Forward mode** needs m forward passes to get a full Jacobian (all gradients of output with respect to each input), where m is the number of inputs

$$\mathbf{y} = f(\mathbf{x}) : \mathbb{R}^m \rightarrow \mathbb{R}^n$$

AutoDiff: Forward Mode

- **Forward mode** needs m forward passes to get a full Jacobian (all gradients of output with respect to each input), where m is the number of inputs

$$\mathbf{y} = f(\mathbf{x}) : \mathbb{R}^m \rightarrow \mathbb{R}^n$$

Problem: DNN typically has large number of inputs:

image as an input, plus all the weights and biases of layers = millions of inputs!

and very few outputs (many DNNs have $n = 1$)

slide adopted from T. Chen, H. Shen, A. Krishnamurthy

AutoDiff: Forward Mode

- **Forward mode** needs m forward passes to get a full Jacobian (all gradients of output with respect to each input), where m is the number of inputs

$$\mathbf{y} = f(\mathbf{x}) : \mathbb{R}^m \rightarrow \mathbb{R}^n$$

Problem: DNN typically has large number of inputs:

image as an input, plus all the weights and biases of layers = millions of inputs!

and very few outputs (many DNNs have $n = 1$)

- Automatic differentiation in **reverse mode** computes all gradients in n backwards passes (so for most DNNs in a single back pass — **back propagation**)

slide adopted from T. Chen, H. Shen, A. Krishnamurthy

Recap: AutoDiff: Forward Mode

- **Forward mode** needs m forward passes to get a full Jacobian (all gradients of output with respect to each input), where m is the number of inputs

$$\mathbf{y} = f(\mathbf{x}) : \mathbb{R}^m \rightarrow \mathbb{R}^n$$

Problem: DNN typically has large number of inputs:

image as an input, plus all the weights and biases of layers = millions of inputs!

and very few outputs (many DNNs have $n = 1$)

slide adopted from T. Chen, H. Shen, A. Krishnamurthy

Recap: AutoDiff: Forward Mode

- **Forward mode** needs m forward passes to get a full Jacobian (all gradients of output with respect to each input), where m is the number of inputs

$$\mathbf{y} = f(\mathbf{x}) : \mathbb{R}^m \rightarrow \mathbb{R}^n$$

Problem: DNN typically has large number of inputs:

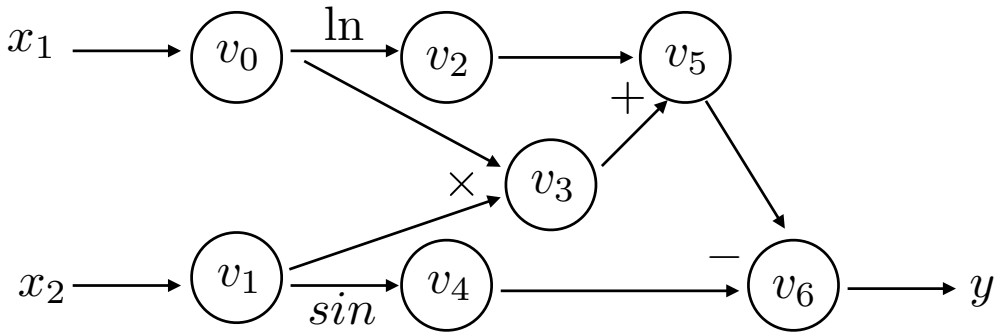
image as an input, plus all the weights and biases of layers = millions of inputs!

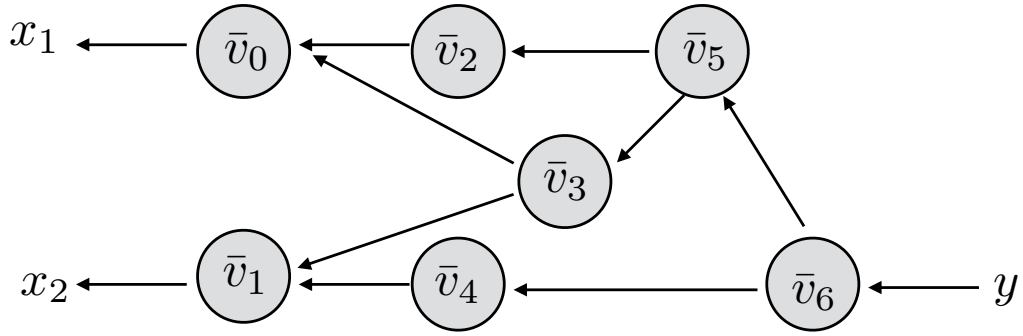
and very few outputs (many DNNs have $n = 1$)

- Automatic differentiation in **reverse mode** computes all gradients in n backwards passes (so for most DNNs in a single back pass — **back propagation**)

slide adopted from T. Chen, H. Shen, A. Krishnamurthy

AutoDiff: Reverse Mode





Forward Evaluation Trace:

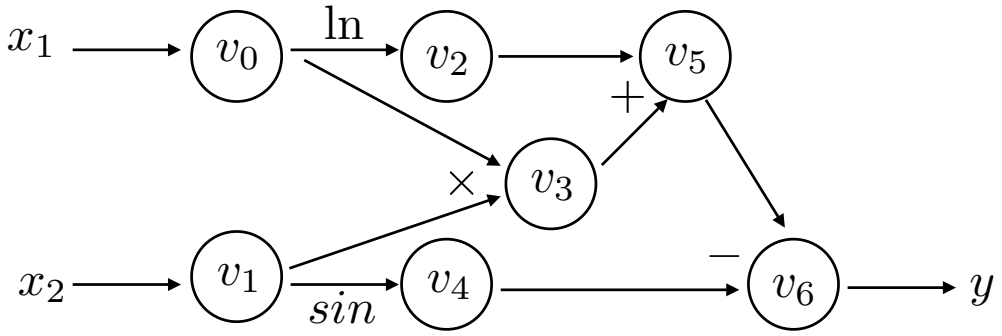
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Traverse the original graph in the reverse topological order and for each node in the original graph introduce an **adjoint node**, which computes derivative of the output with respect to the local node (using Chain rule):

$$\bar{v}_i = \frac{\partial y_j}{\partial v_i} = \sum_{k \in \mathrm{pa}(i)} \frac{\partial v_k}{\partial v_i} \frac{\partial y_j}{\partial v_k} = \sum_{k \in \mathrm{pa}(i)} \frac{\partial v_k}{\partial v_i} \bar{v}_k$$

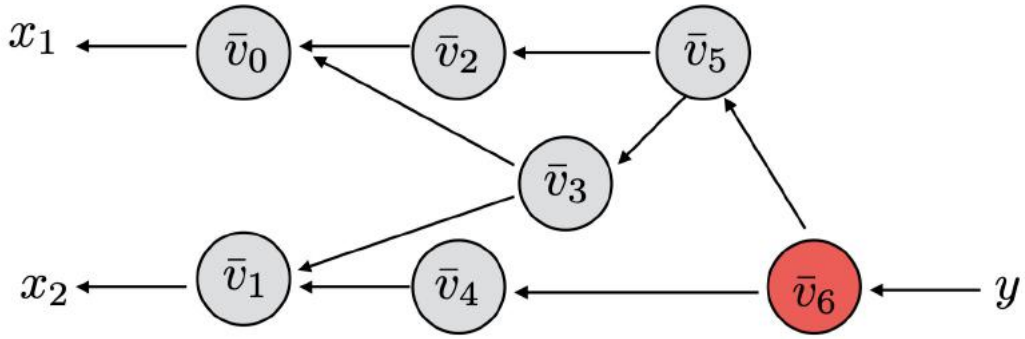
"local" derivative

AutoDiff: Reverse Mode



Forward Evaluation Trace:

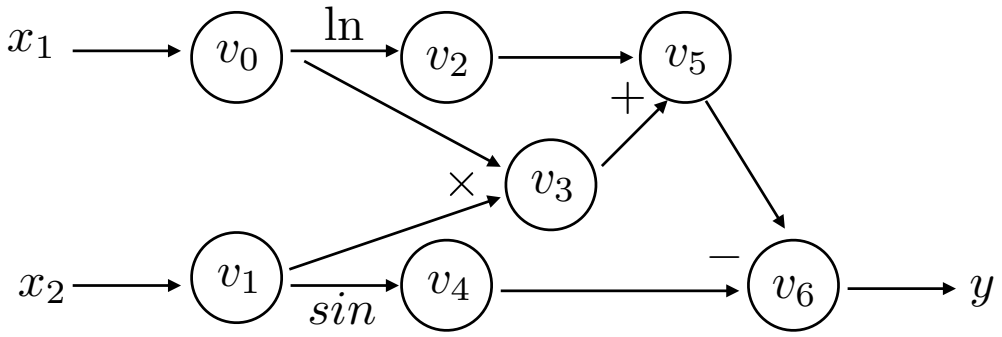
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

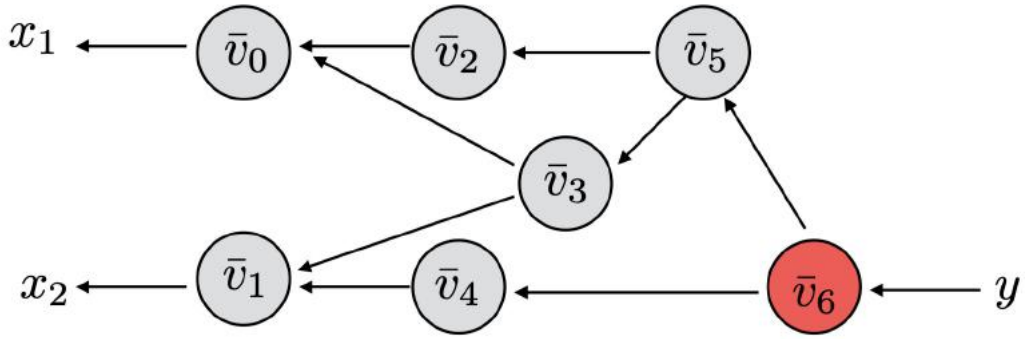


Backwards Derivative Trace:

$$\bar{v}_6 = \frac{\partial y}{\partial v_6}$$

AutoDiff: Reverse Mode





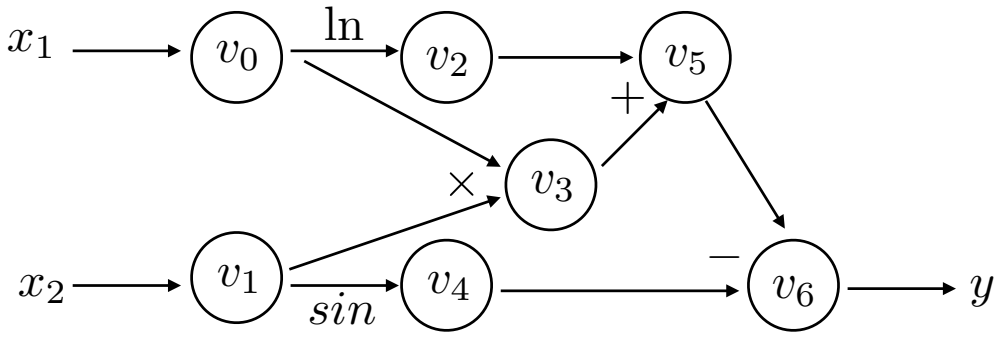
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

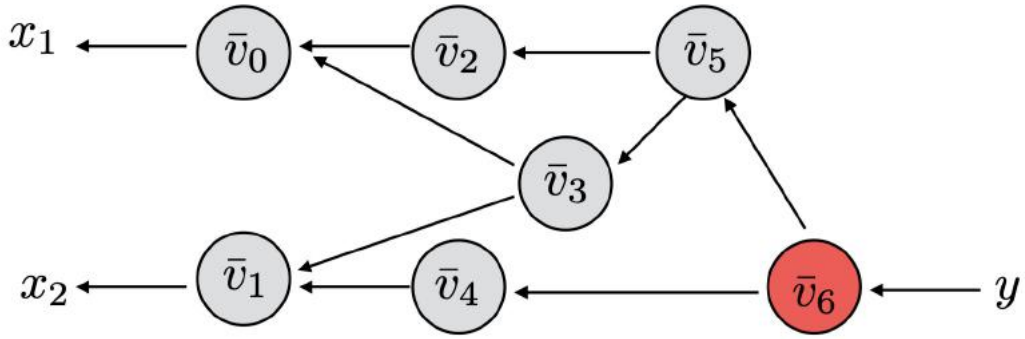
$$\bar{v}_6 = \frac{\partial y}{\partial v_6}$$

AutoDiff: Reverse Mode



Forward Evaluation Trace:

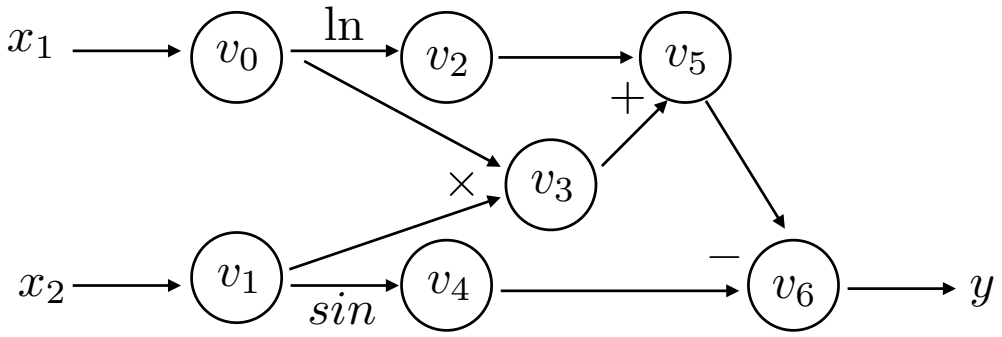
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652



Backwards Derivative Trace:

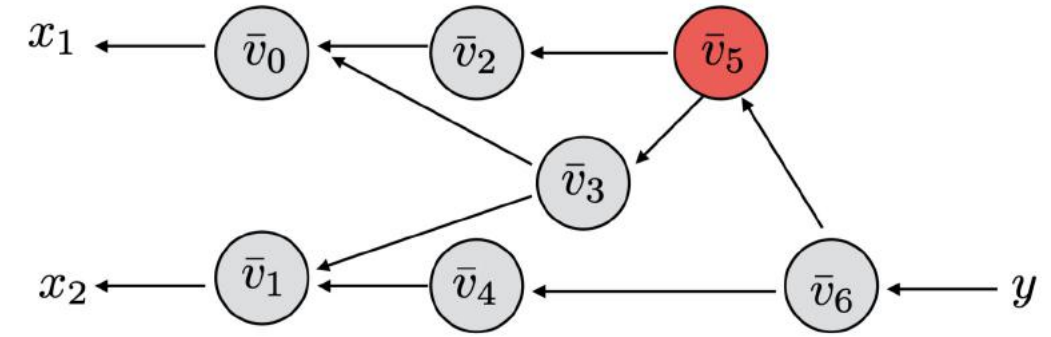
$\bar{v}_6 = \dfrac{\partial y}{\partial v_6}$	1

AutoDiff: Reverse Mode



Forward Evaluation Trace:

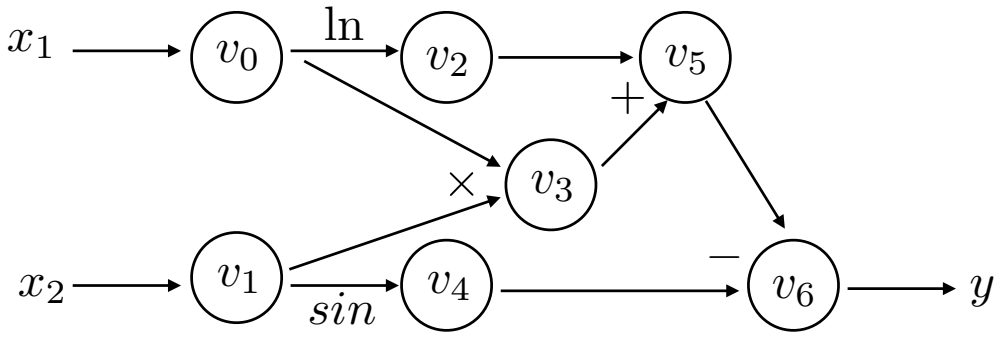
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

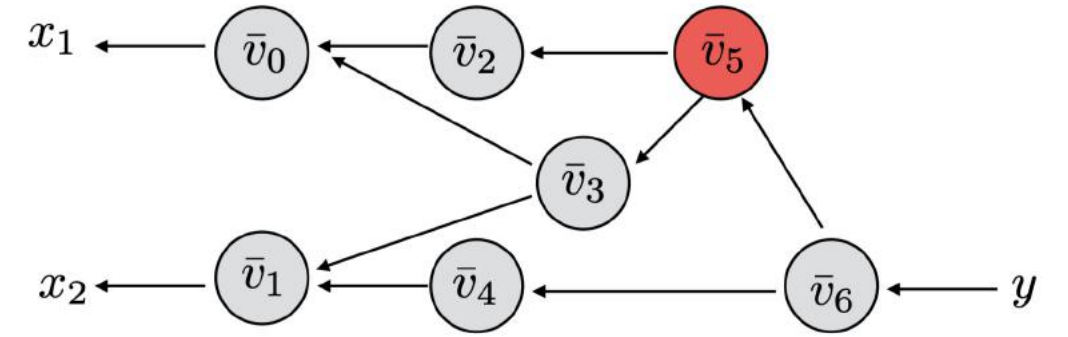


Backwards Derivative Trace:

$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5}$	
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff: Reverse Mode





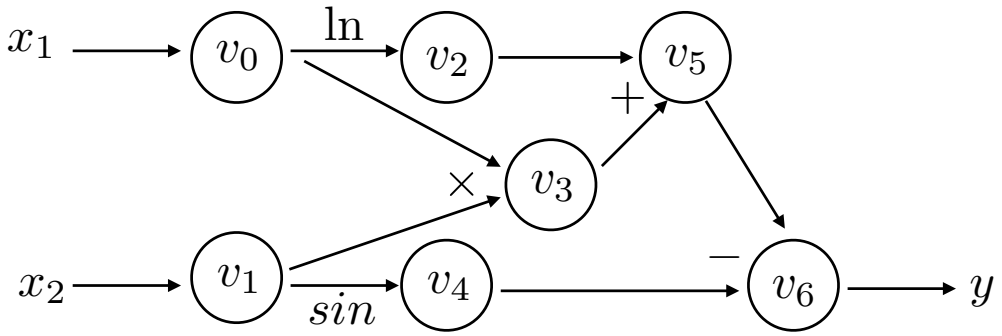
Forward Evaluation Trace:

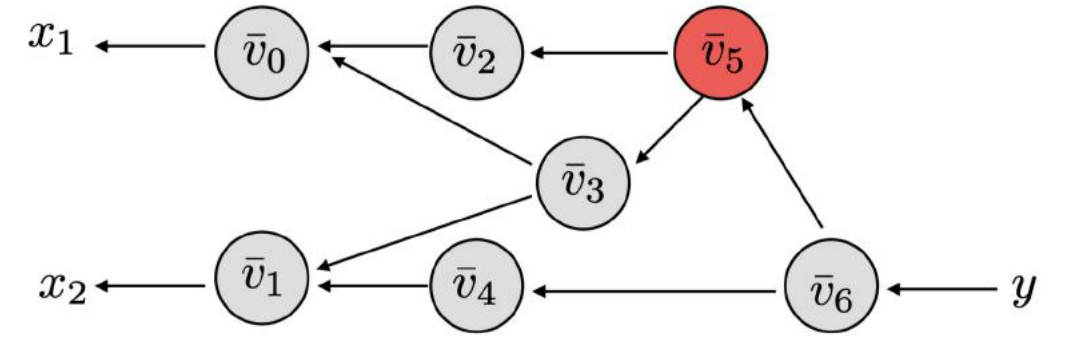
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5}$	
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff: Reverse Mode





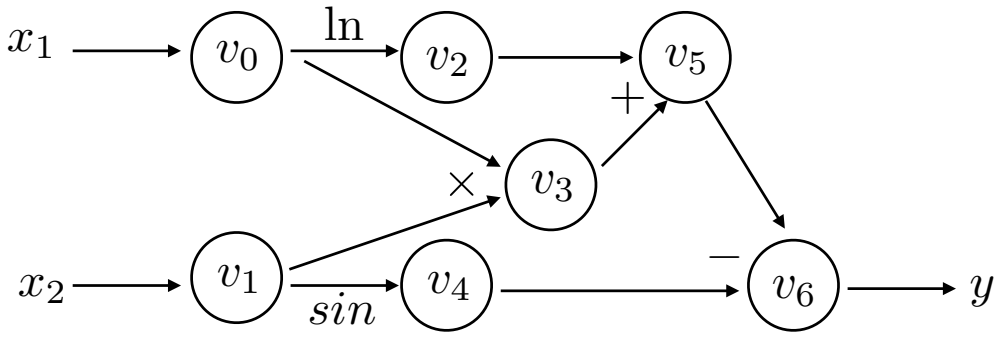
Forward Evaluation Trace:

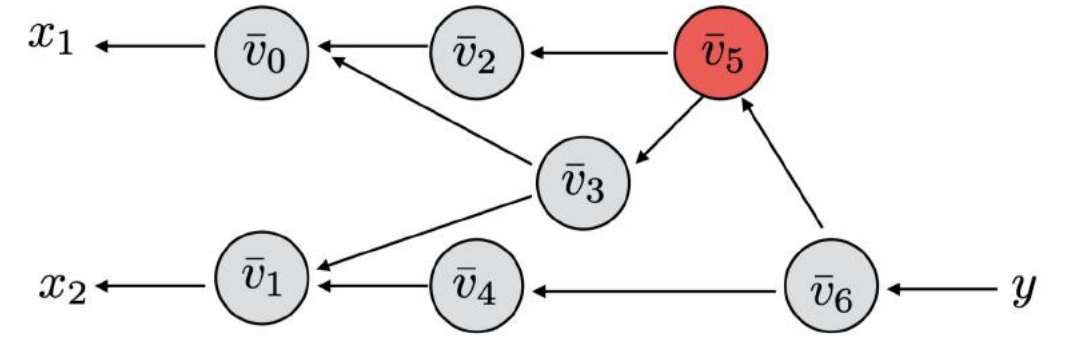
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff: Reverse Mode





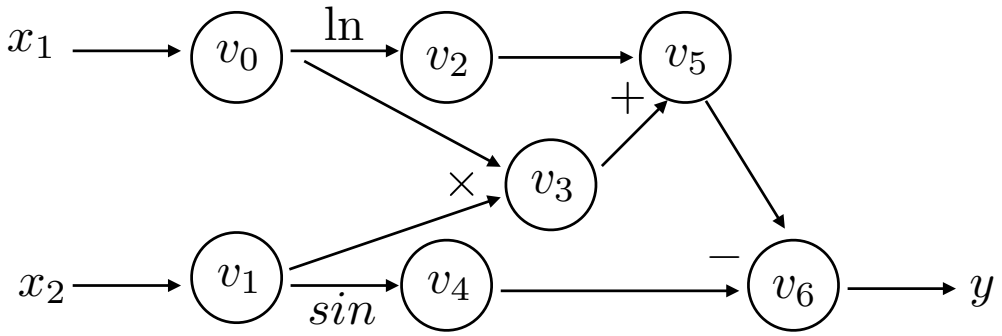
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

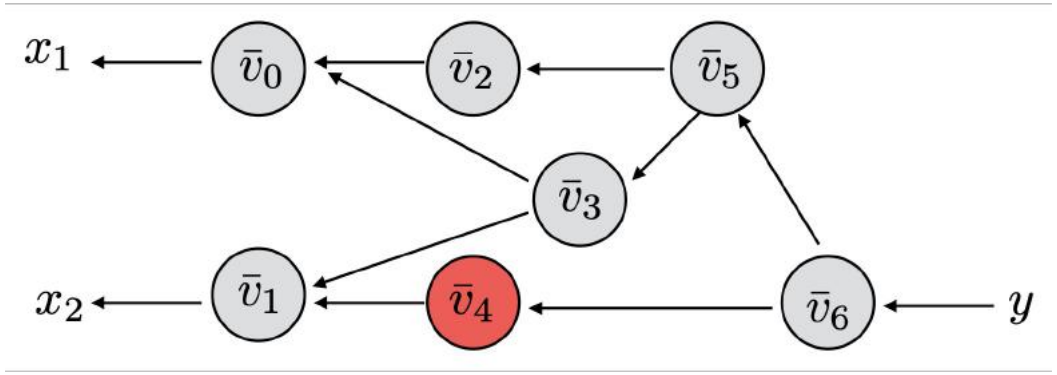
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff: Reverse Mode



Forward Evaluation Trace:

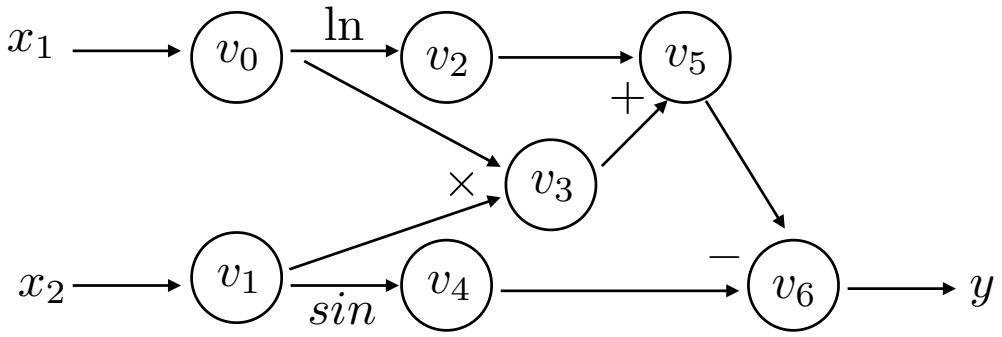
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652



Backwards Derivative Trace:

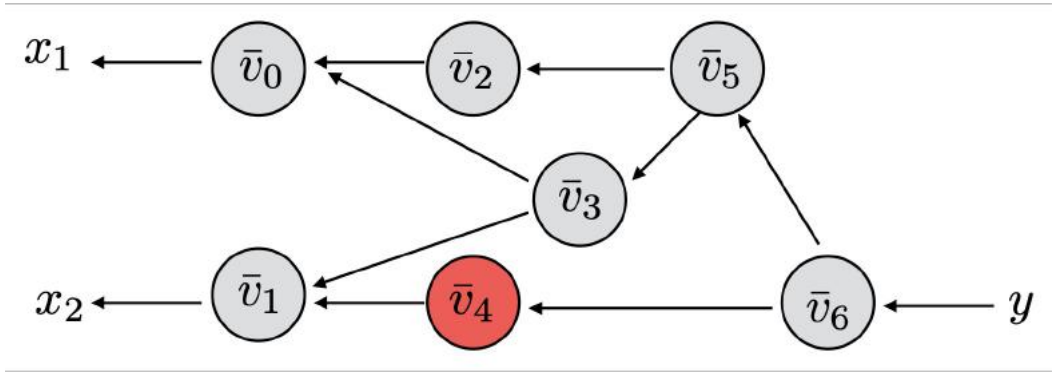
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4}$	
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff: Reverse Mode



Forward Evaluation Trace:

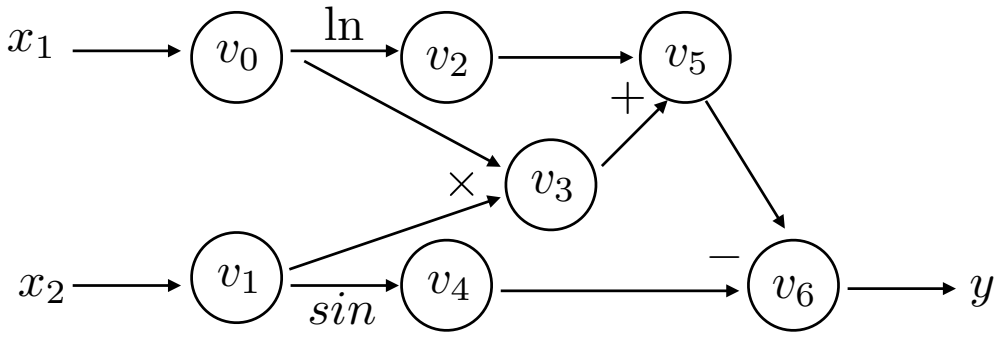
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

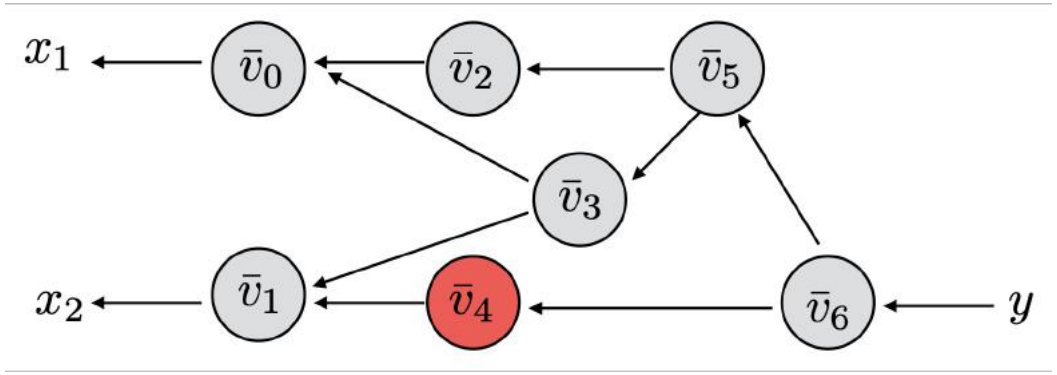


Backwards Derivative Trace:

$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4}$	
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff: Reverse Mode





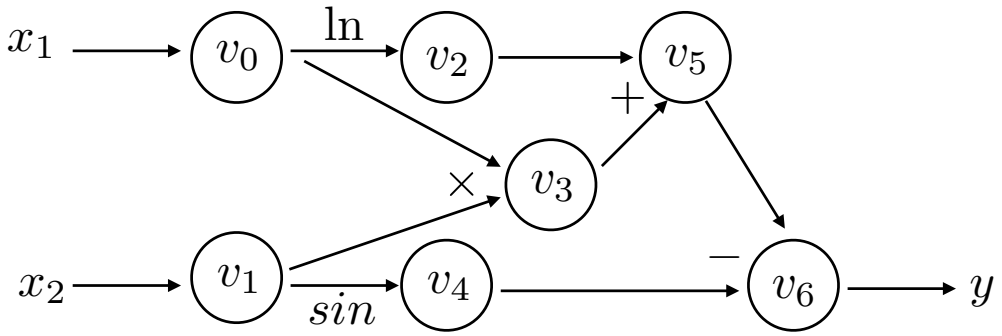
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

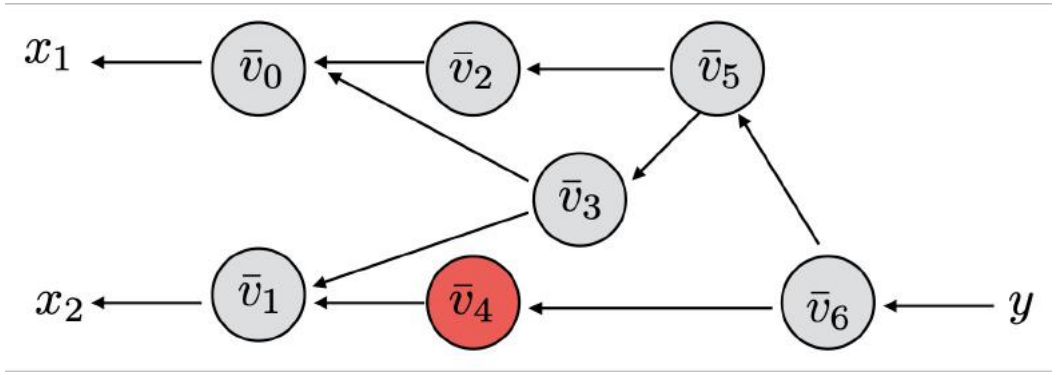
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff: Reverse Mode



Forward Evaluation Trace:

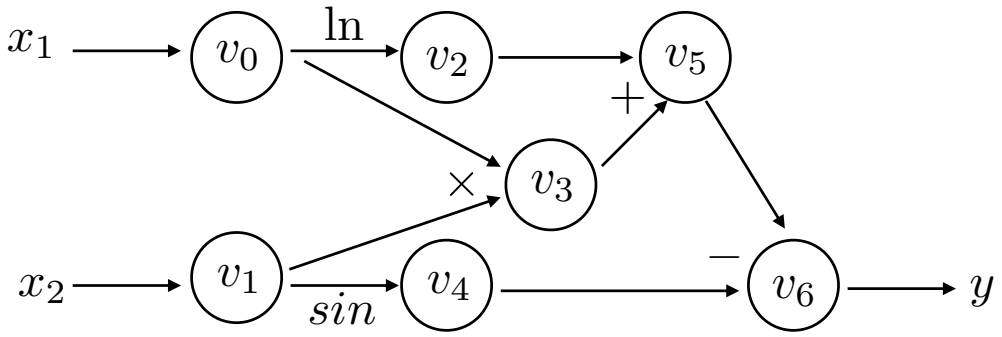
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

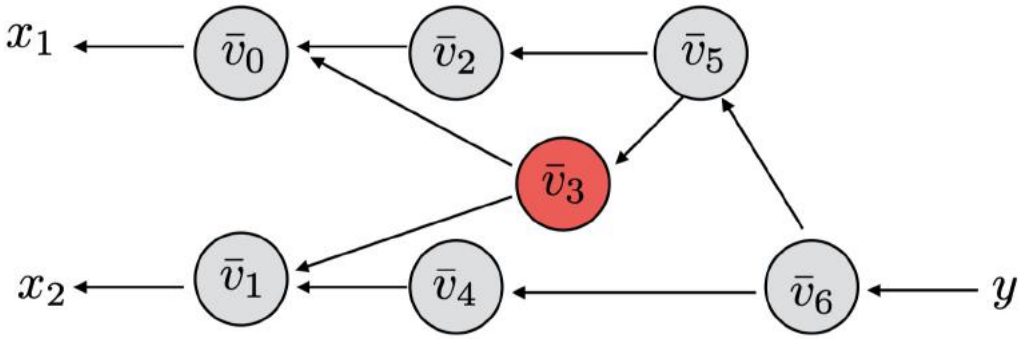


Backwards Derivative Trace:

$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff: Reverse Mode





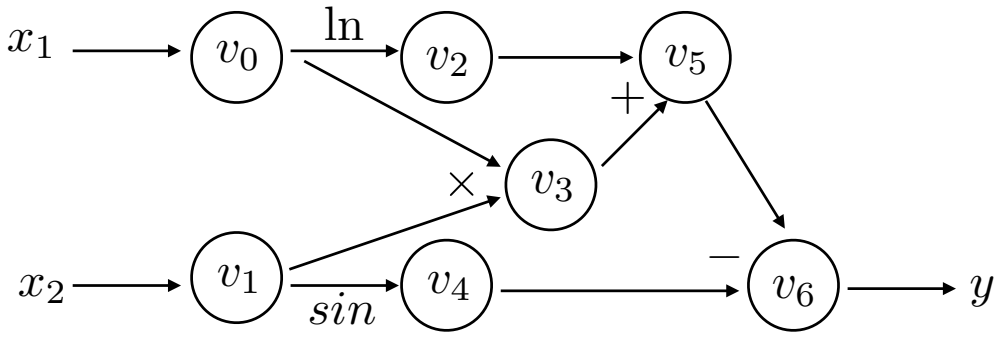
Forward Evaluation Trace:

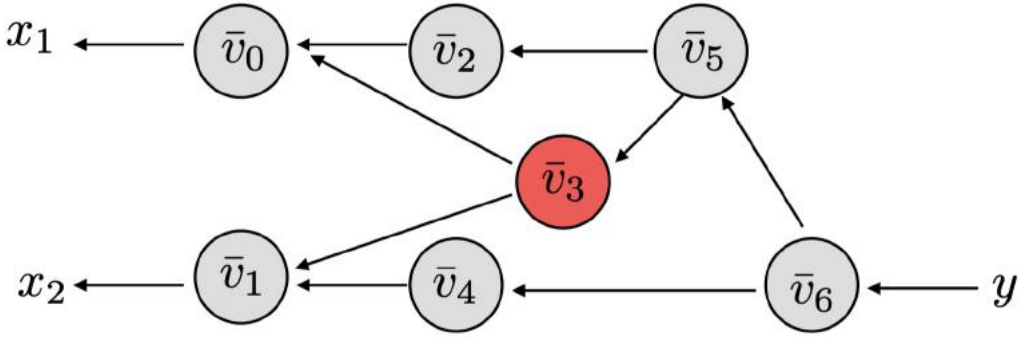
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3}$	
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff: Reverse Mode





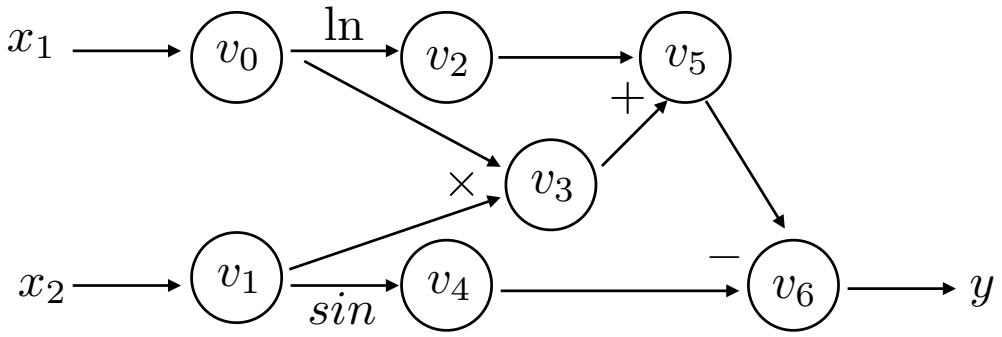
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

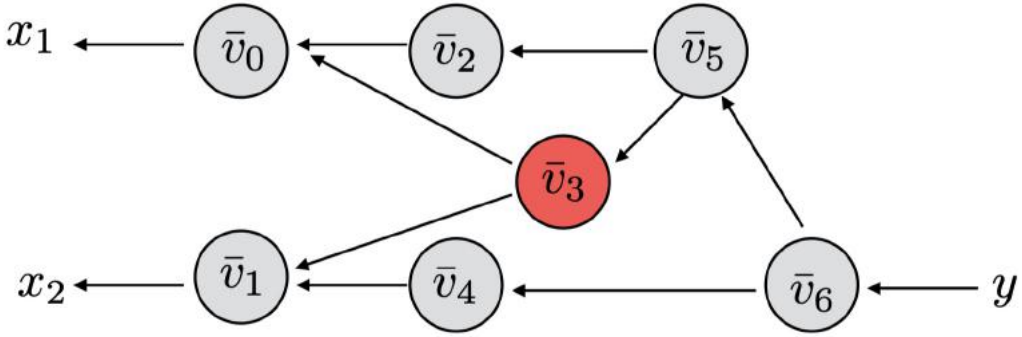
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3}$	
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff: Reverse Mode



Forward Evaluation Trace:

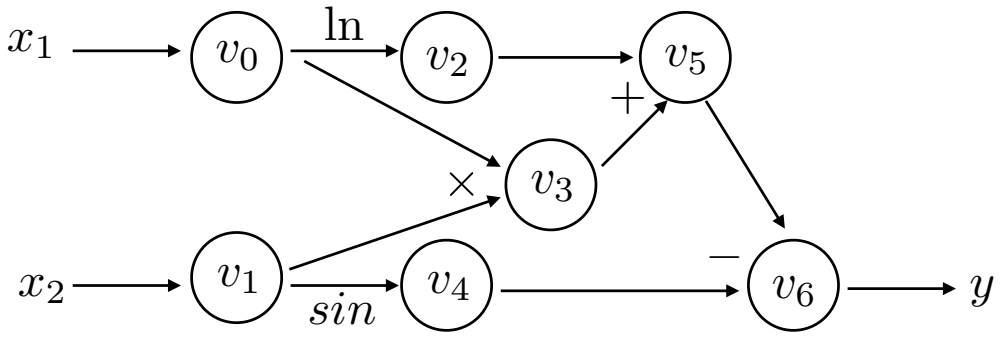
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

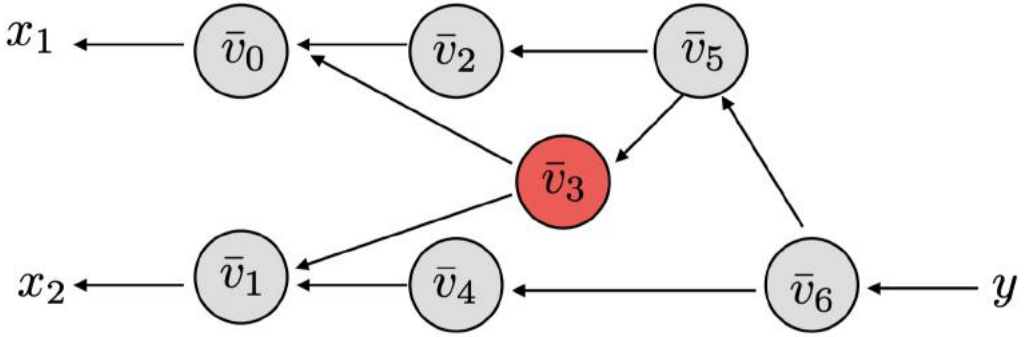


Backwards Derivative Trace:

$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff: Reverse Mode





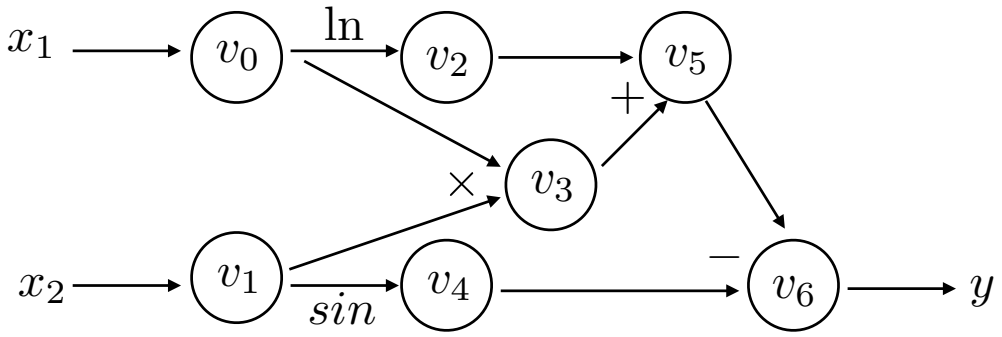
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

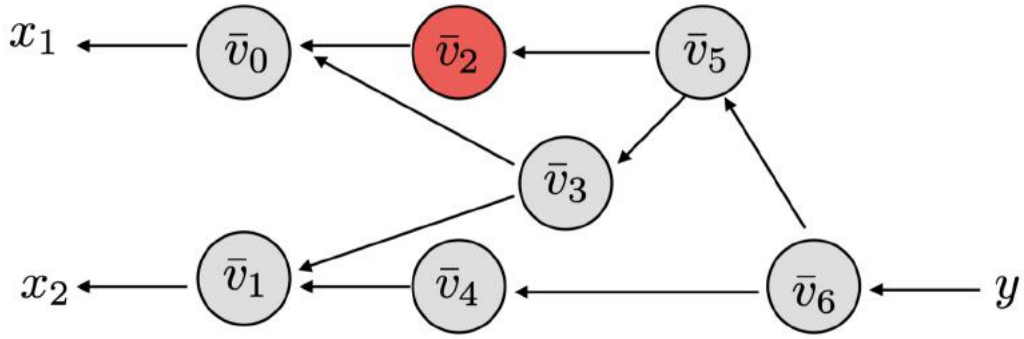
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff: Reverse Mode



Forward Evaluation Trace:

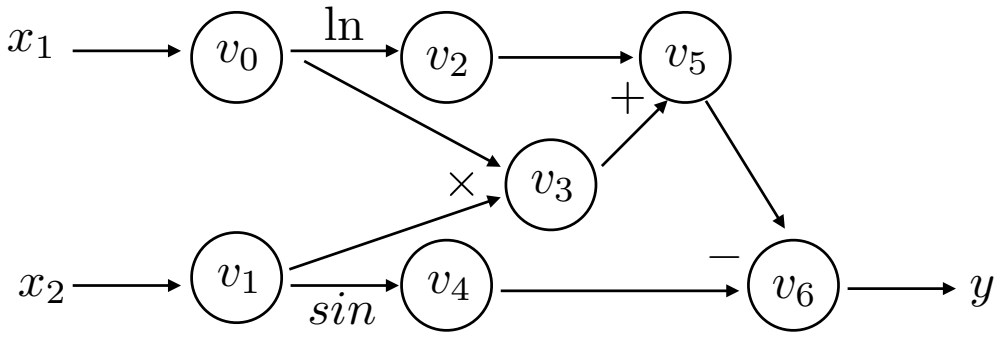
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652



Backwards Derivative Trace:

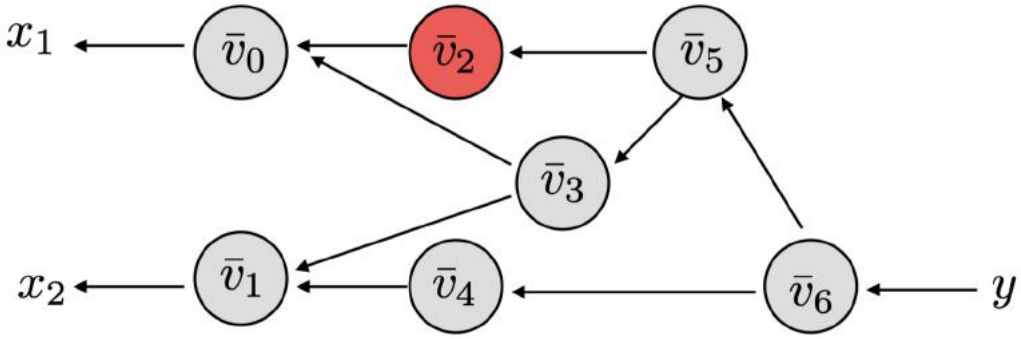
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2}$	
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff: Reverse Mode



Forward Evaluation Trace:

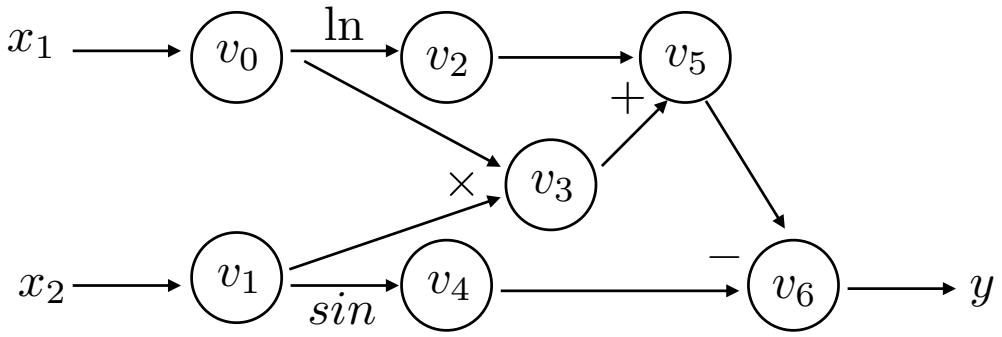
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652



Backwards Derivative Trace:

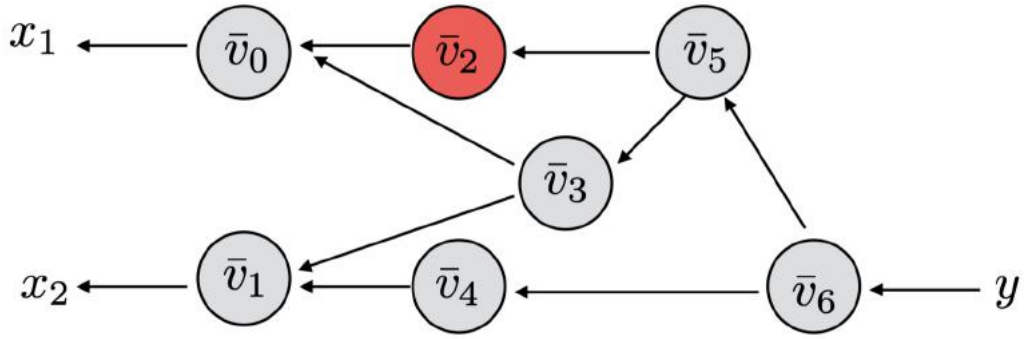
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff: Reverse Mode



Forward Evaluation Trace:

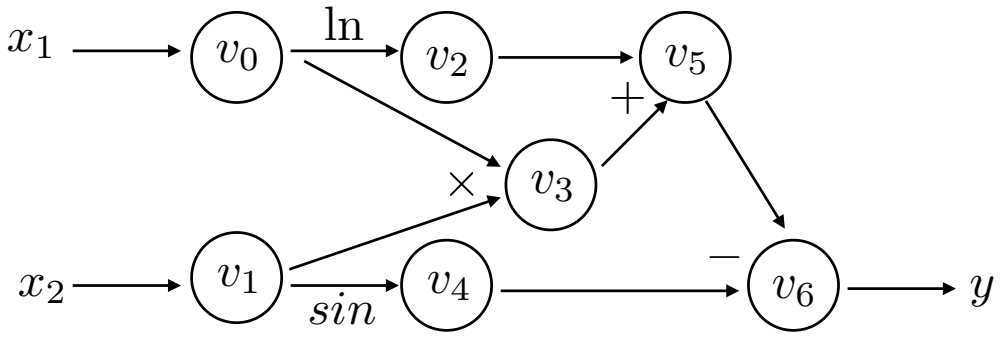
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652



Backwards Derivative Trace:

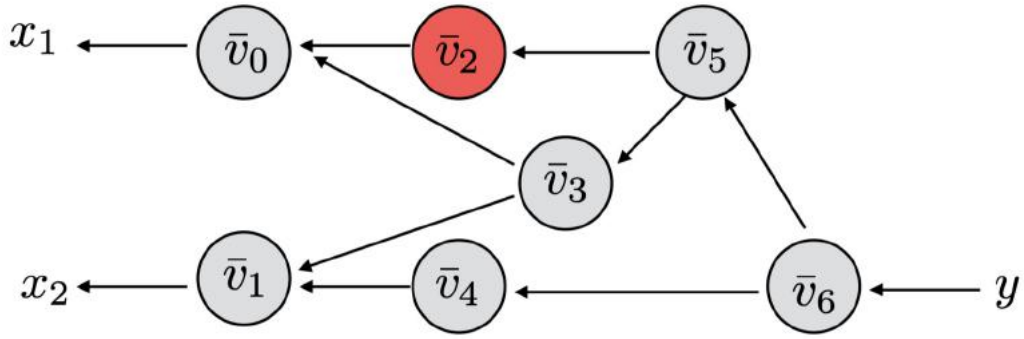
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff: Reverse Mode



Forward Evaluation Trace:

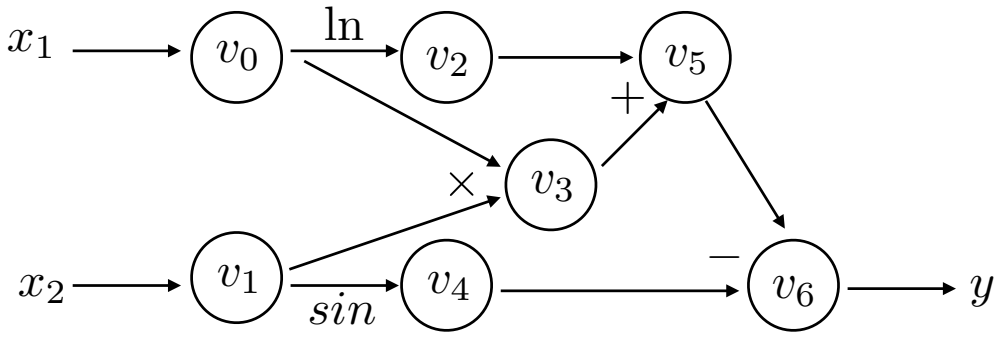
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

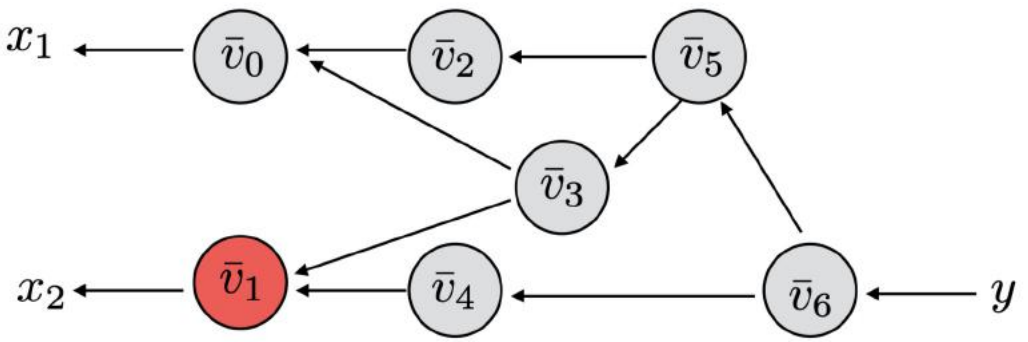


Backwards Derivative Trace:

$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff: Reverse Mode





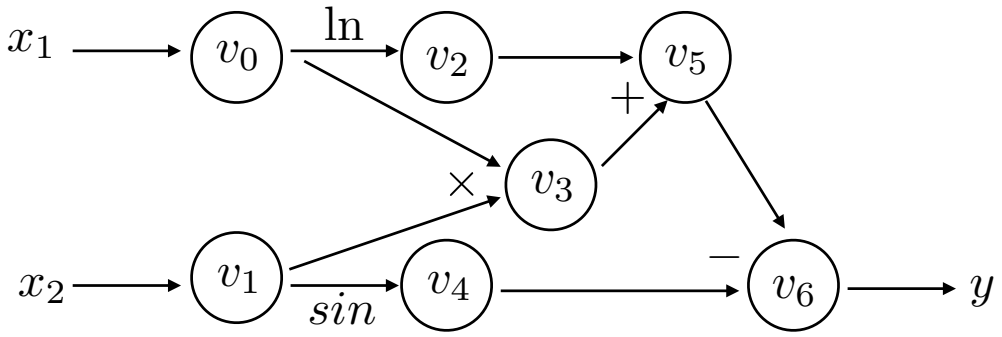
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

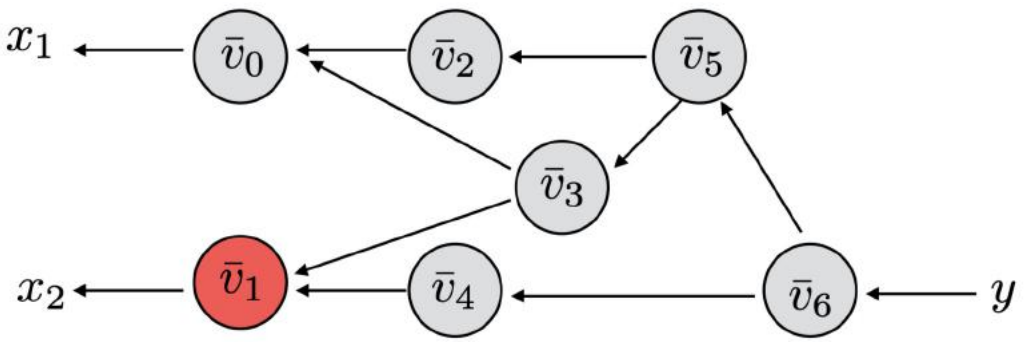
$\bar{v}_1$	
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff: Reverse Mode



Forward Evaluation Trace:

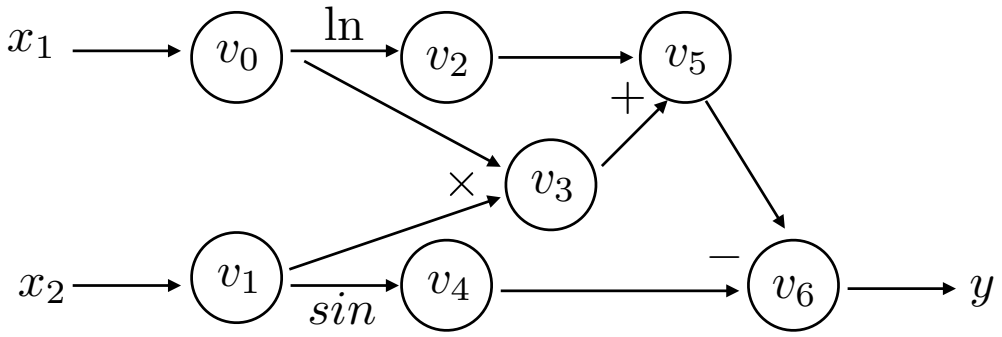
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652



Backwards Derivative Trace:

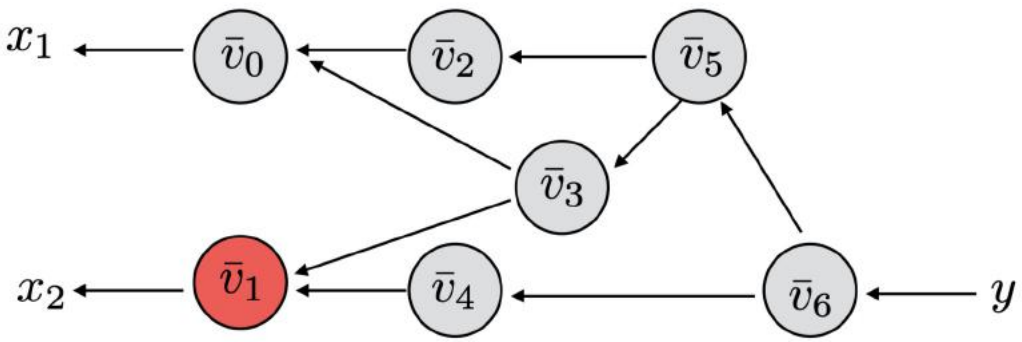
$\bar{v}_1 = \bar{v}_3 \frac{\partial v_3}{\partial v_1} + \bar{v}_4 \frac{\partial v_4}{\partial v_1}$	
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff: Reverse Mode



Forward Evaluation Trace:

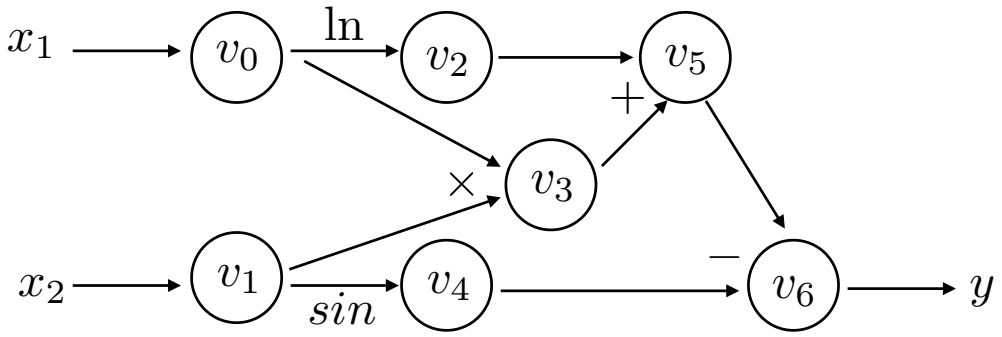
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

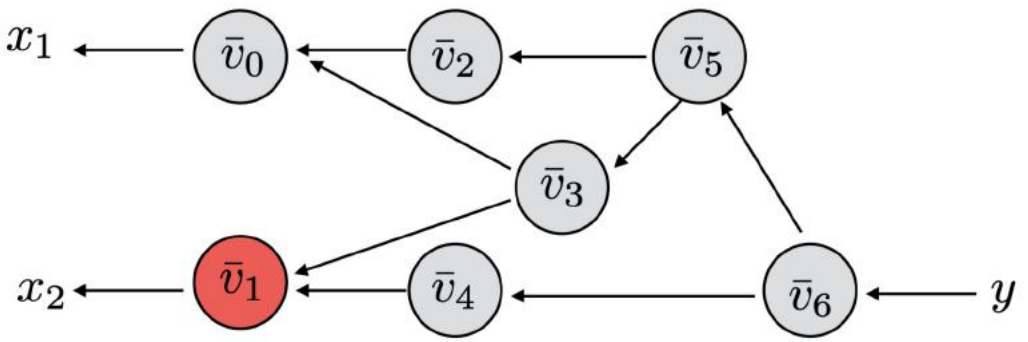


Backwards Derivative Trace:

$\bar{v}_1 = \bar{v}_3 \frac{\partial v_3}{\partial v_1} + \bar{v}_4 \frac{\partial v_4}{\partial v_1}$	
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff: Reverse Mode





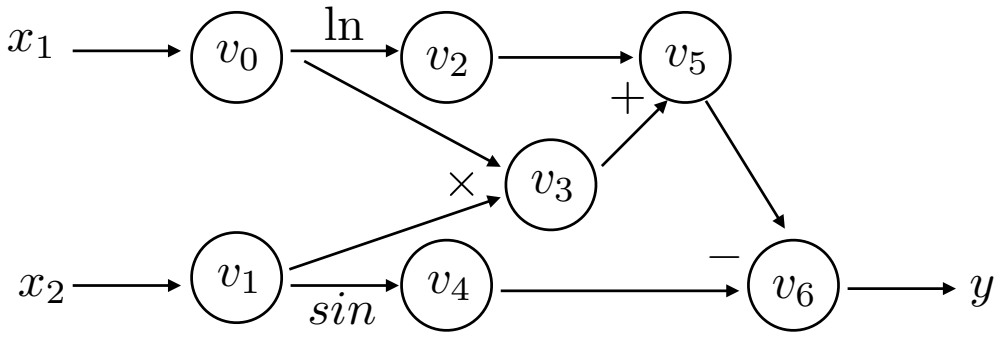
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

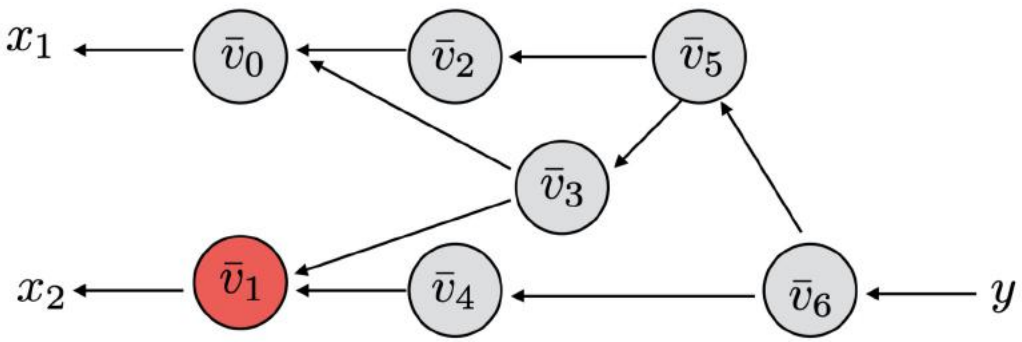
$\bar{v}_1 = \bar{v}_3 \frac{\partial v_3}{\partial v_1} + \bar{v}_4 \frac{\partial v_4}{\partial v_1} = \bar{v}_3 v_0 + \bar{v}_4 cos(v_1)$	
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff: Reverse Mode



Forward Evaluation Trace:

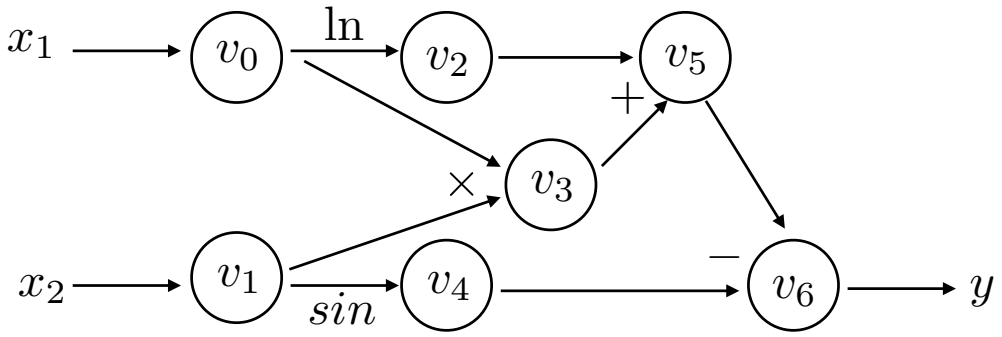
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

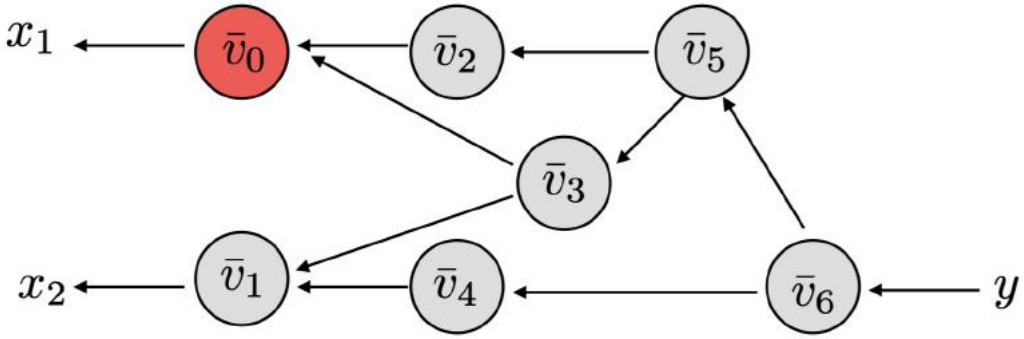


Backwards Derivative Trace:

$\bar{v}_1 = \bar{v}_3 \frac{\partial v_3}{\partial v_1} + \bar{v}_4 \frac{\partial v_4}{\partial v_1} = \bar{v}_3 v_0 + \bar{v}_4 cos(v_1)$	1.716
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff: Reverse Mode





Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = -0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

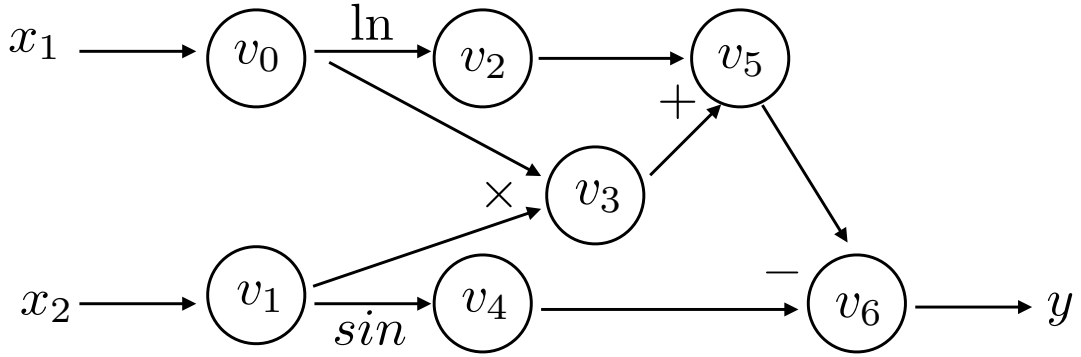
$\bar{v}_0 = \bar{v}_3 \frac{\partial v_3}{\partial v_0} + \bar{v}_2 \frac{\partial v_2}{\partial v_0} = \bar{v}_3 v_1 + \bar{v}_2 \frac{1}{v_0}$	5.5
$\bar{v}_1 = \bar{v}_3 \frac{\partial v_3}{\partial v_1} + \bar{v}_4 \frac{\partial v_4}{\partial v_1} = \bar{v}_3 v_0 + \bar{v}_4 cos(v_1)$	1.716
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

Automatic Differentiation (AutoDiff)

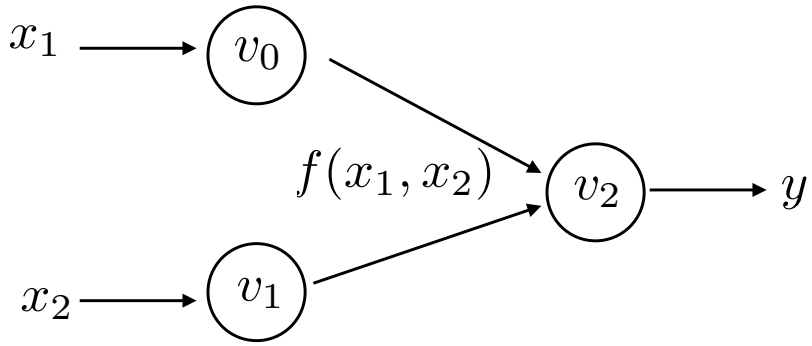
$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

- AutoDiff can be done at various **granularities**

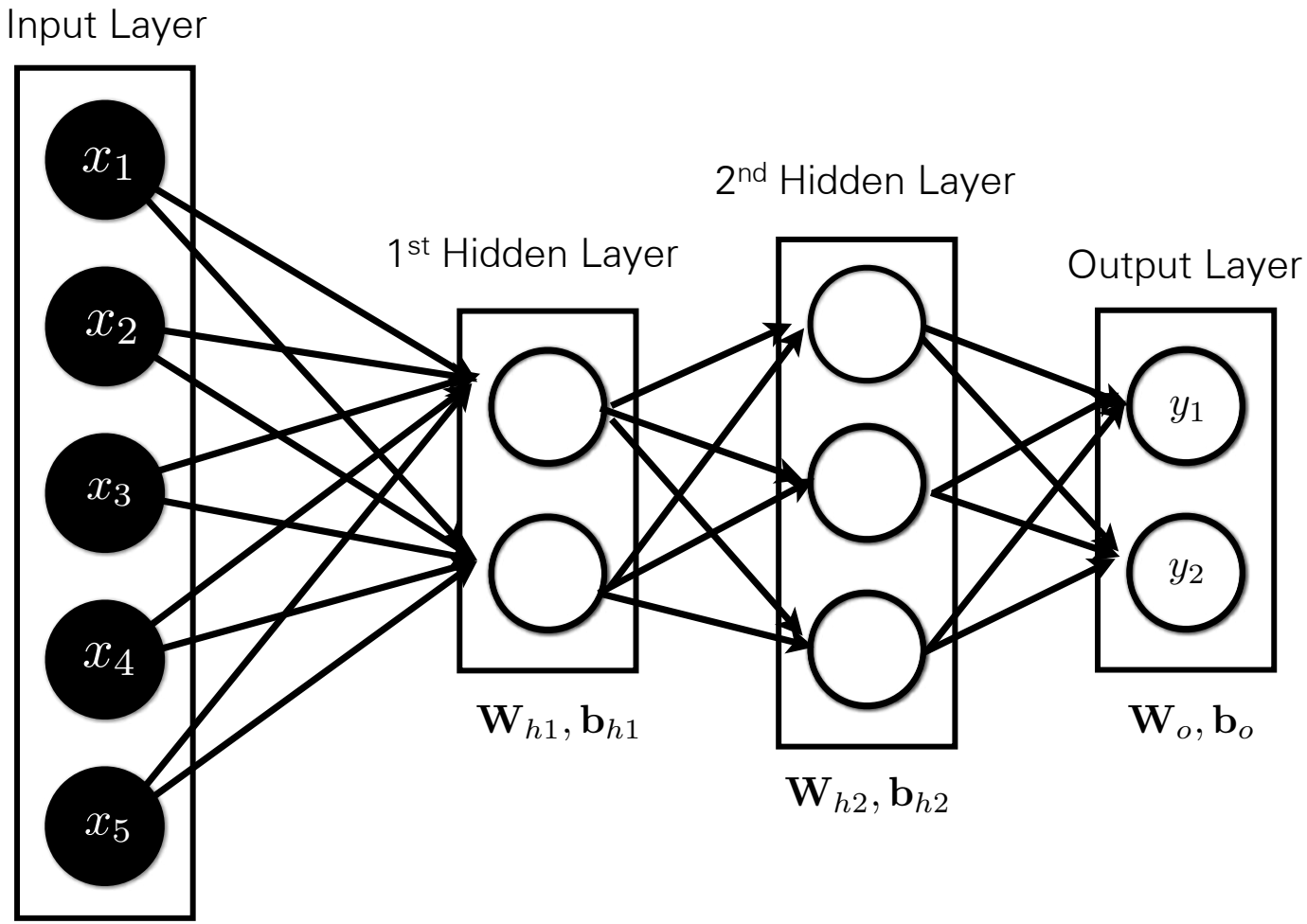
Elementary function granularity:



Complex function granularity:



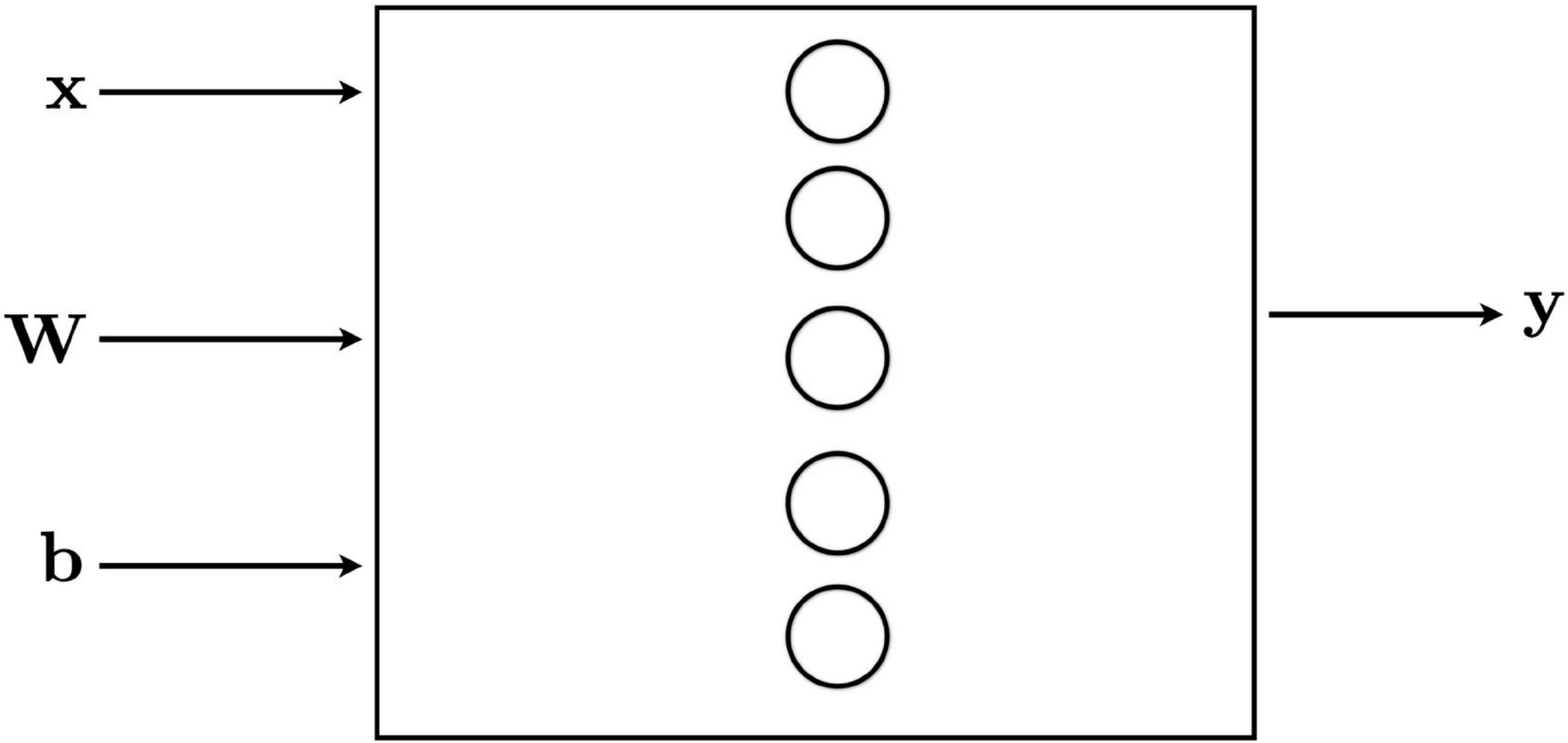
Backpropagation: Practical Issues



Easier to deal with in vector form

Backpropagation: Practical Issues

$$\mathbf{y} = f(\mathbf{W}, \mathbf{b}, \mathbf{x}) = \mathbf{sigmoid}(\mathbf{W} \cdot \mathbf{x} + \mathbf{b})$$

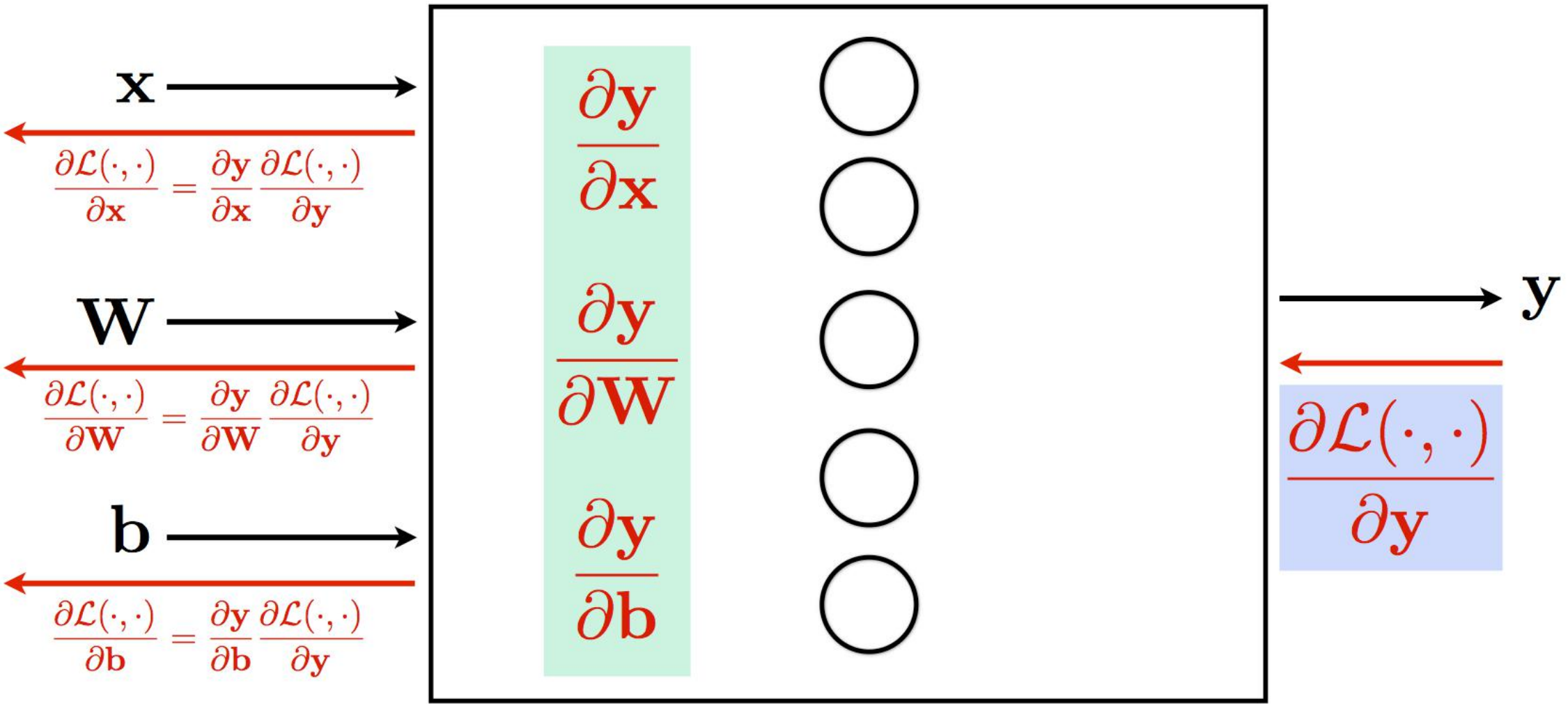


Backpropagation: Practical Issues

"local" Jacobians
(matrix of partial derivatives,
e.g. |x| x |y|

$$\mathbf{y} = f(\mathbf{W}, \mathbf{b}, \mathbf{x}) = \mathbf{sigmoid}(\mathbf{W} \cdot \mathbf{x} + \mathbf{b})$$

"backprop" Gradient



Jacobian of Sigmoid layer

$\mathbf{x}, \mathbf{y} \in \mathbb{R}^{2048}$

- Element-wise sigmoid layer:

Jacobian of Sigmoid layer

$\mathbf{x}, \mathbf{y} \in \mathbb{R}^{2048}$

- Element-wise sigmoid layer:

  - What is the dimension of Jacobian?

Jacobian of Sigmoid layer

$\mathbf{x}, \mathbf{y} \in \mathbb{R}^{2048}$

- Element-wise sigmoid layer:

  - What is the dimension of Jacobian?
  - What does it look like?

Jacobian of Sigmoid layer

$\mathbf{x}, \mathbf{y} \in \mathbb{R}^{2048}$

- Element-wise sigmoid layer:

  - What is the dimension of **Jacobian**?
  - What does it look like?

    If we are working with a mini batch of 100 inputs-output pairs, Jacobian is a matrix 204,800 x 204,800!

Backpropagation: Common questions

- **Question:** Does BackProp only work for certain layers?

  Answer: No, for any differentiable functions

- **Question:** What is computational cost of BackProp?

  Answer: On average about twice the forward pass

- **Question:** Is BackProp a dual of forward propagation?

  Answer: Yes

Backpropagation: Common questions

- **Question:** Does BackProp only work for certain layers?

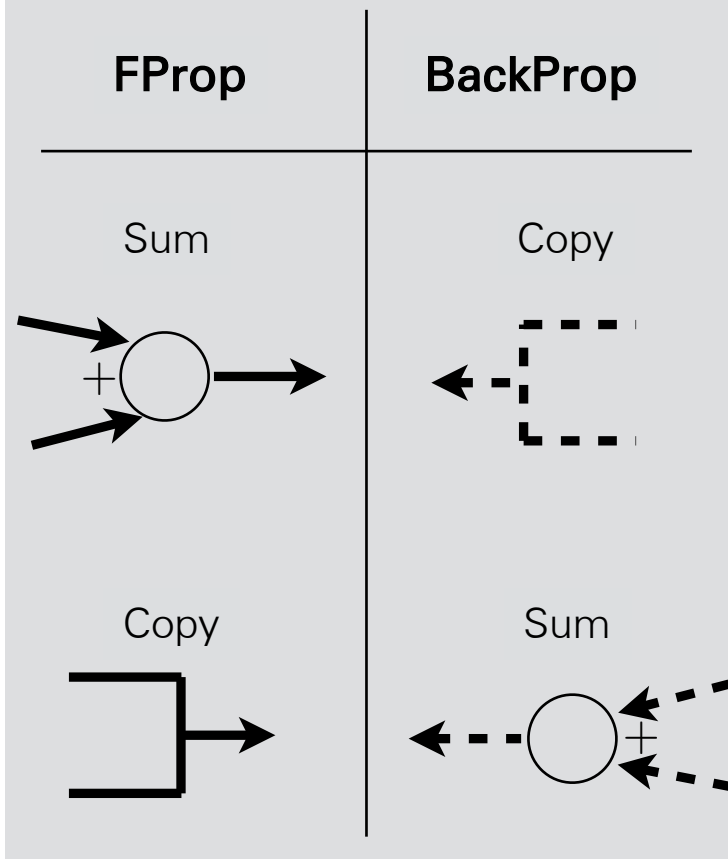
  Answer: No, for any differentiable functions

- **Question:** What is computational cost of BackProp?

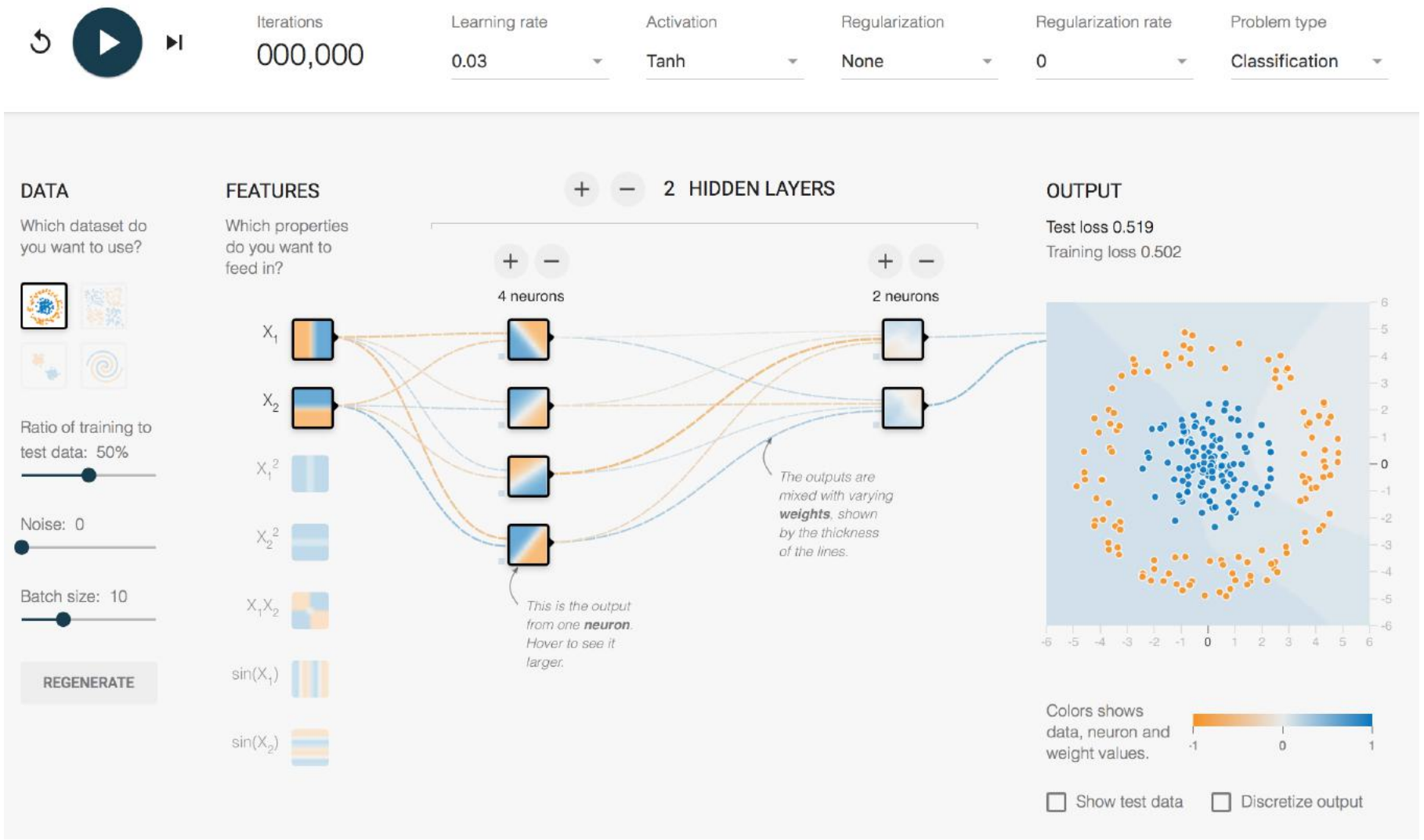
  Answer: On average about twice the forward pass

- **Question:** Is BackProp a dual of forward propagation?

  Answer: Yes



Demo time

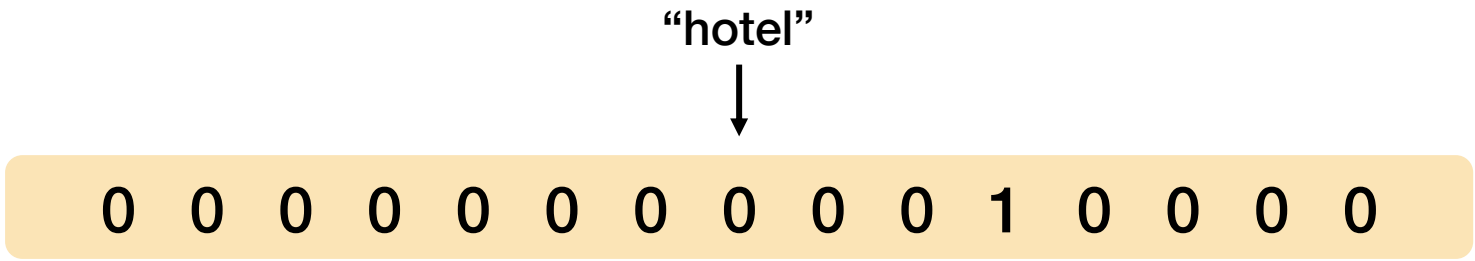


http://playground.tensorflow.org

Shallow yet very powerful: `word2vec`

From symbolic to distributed word representations

- The vast majority of rule-based or statistical NLP and IR work regarded words as atomic symbols: **hotel**, **conference**, **walk**
- In vector space terms, this is a vector with one 1 and a lot of zeroes



- We now call this a **one-hot** representation.

From symbolic to distributed word representations

- The size of word vectors are equal to the number of words in the dictionary
  - Vector size is proportional to the size of the dictionary
    20K (speech) – 50K (Pen Treebank) – 500K (A large dictionary) – 13M (Google 1T)
- One-hot vectors vectors are **orthogonal**
- There is no natural notion of similarity in a set of one-hot vectors

“motel”	**0 0 0 0 0 0 0 0 0 1 0 0 0 0 0**	**T**
“hotel”	**0 0 0 0 0 0 0 0 0 0 1 0 0 0 0**	**= 0**

Distributional similarity-based representations

- You can get a lot of value by representing a word by means of its neighbors

- "You shall know a word by the company it keeps"

(J. R. Firth 1957:11)

- One of the most successful ideas of modern NLP

government debt problems turning into	banking	crises as has happened in
saying that Europe needs unified	banking	regulation to replace the hodgepodge

These words will represent **"banking"**

Distributional hypothesis

- The meaning of a word is (can be approximated by, derived from) the set of contexts in which it occurs in texts

He filled the **wampimuk**, passed it around and we all drunk some

We found a little, hairy **wampimuk** sleeping behind the tree

Testing the distributional hypothesis: The influence of context on judgements of semantic similarity [McDonald & Ramscar'01]

Slide credit: Marco Baroni

Distributional semantics

he curtains open and the moon shining in on the barely
ars and the cold , close moon " . And neither of the w
rough the night with the moon shining so brightly , it
made in the light of the moon . It all boils down , wr
surely under a crescent moon , thrilled by ice-white
sun , the seasons of the moon ? Home , alone , Jay pla
m is dazzling snow , the moon has risen full and cold
un and the temple of the moon , driving out of the hug
in the dark and now the moon rises , full and amber a
bird on the shape of the moon over the trees in front
But I could n't see the moon or the stars , only the
rning , with a sliver of moon hanging among the stars
they love the sun , the moon and the stars . None of
the light of an enormous moon . The plash of flowing w
man 's first step on the moon ; various exhibits , aer
the inevitable piece of moon rock . Housing The Airsh
oud obscured part of the moon . The Allied guns behind

A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge [Landauer and Dumais'97]
From frequency to meaning: Vector space models of semantics [Turney ve Pantel'10]
…

Slide credit: Marco Baroni

Window based co-occurence matrix

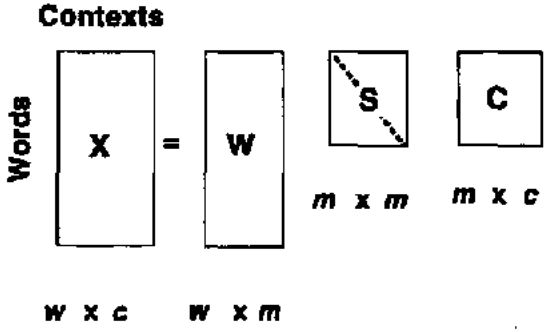
- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

- Increase in size with vocabulary
- Very high dimensional: require a lot of storage
- Subsequent classification models have sparsity issues
- Models are less robust

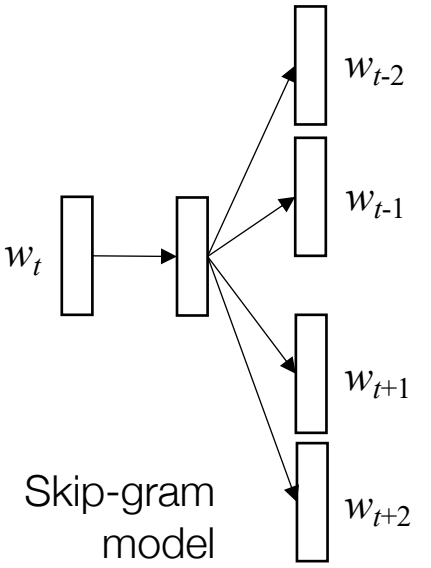
Three methods for getting short dense vectors

- Singular Value Decomposition of co-occurrence matrix X
  - A special case of this is called LSA – Latent Semantic Analysis



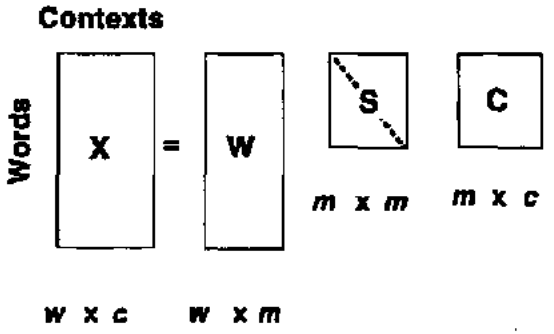
- Neural Language Model-inspired predictive models
  - skip-grams and CBOW

- Brown clustering



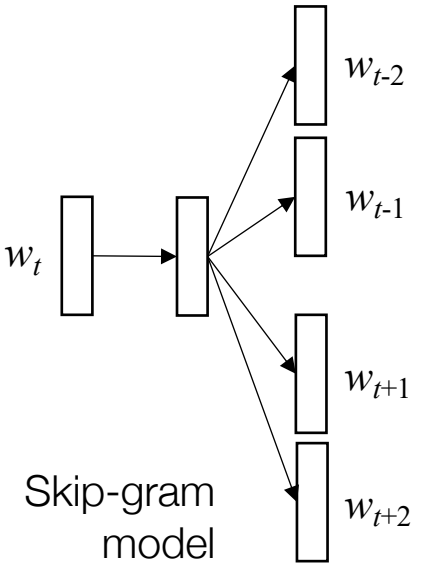
Three methods for getting short dense vectors

- Singular Value Decomposition of co-occurrence matrix X
  - A special case of this is called LSA – Latent Semantic Analysis



- Neural Language Model-inspired predictive models
  - skip-grams and CBOW

- Brown clustering



Prediction-based models: An alternative way to get dense vectors

- **Skip-gram** (Mikolov et al. 2013a), **CBOW** (Mikolov et al. 2013b)
- Learn embeddings as part of the process of word prediction.
- Train a neural network to predict neighboring words
  - Inspired by **neural net language models**.
  - In so doing, learn dense embeddings for the words in the training corpus.
- Advantages:
  - Fast, easy to train (much faster than SVD)
  - Available online in the `word2vec` package
  - Including sets of pretrained embeddings!

Basic idea of learning neural network word embeddings

- We define some model that aims to predict a word based on other words in its context

$$\text{argmax}_w w \cdot ((w_{j-1} + w_{j+1})/2)$$

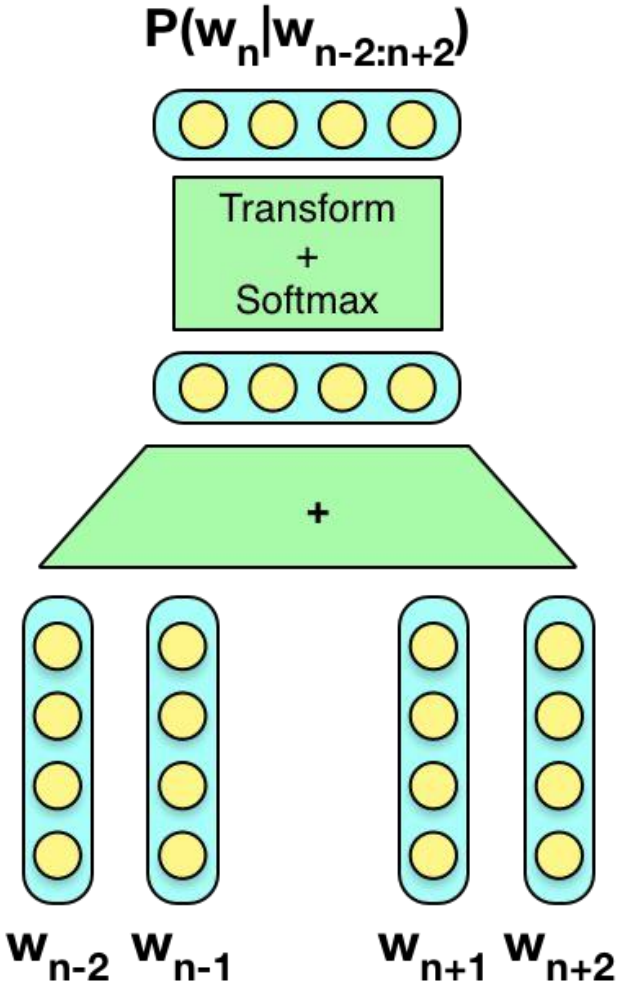
which has a loss function, e.g.,

$$J(\theta) = 1 - w_j \cdot ((w_{j-1} + w_{j+1})/2)$$

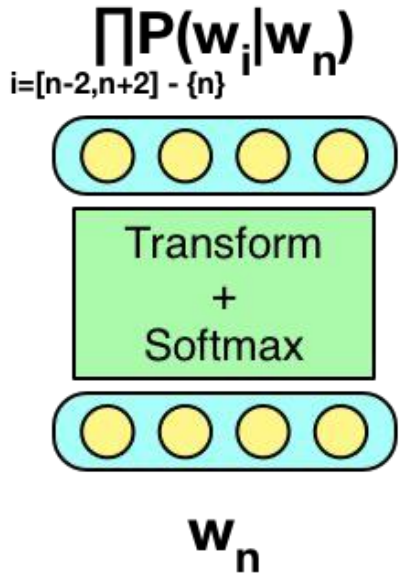
Unit norm vectors

- We look at many samples from a big language corpus
- We keep adjusting the vector representations of words to minimize this loss

Neural Embedding Models (Mikolov et al. 2013)



CBoW model



Skip-gram model

Distributed representations of words and phrases and their compositionality [Mikolov vd.'13]

Image credit: Ed Grefenstette

Details of word2vec

- Predict surrounding words in a window of length m of every word.
- **Objective function:** Maximize the log probability of any context word given the current center word:

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log p(w_{t+j}|w_t)$$

where θ represents all variables we optimize

Distributed representations of words and phrases and their compositionality [Mikolov vd.'13]

Details of word2vec

- Predict surrounding words in a window of length m of every word.
- For $p(w_{t+j}|w_t)$ the simplest first formulation is

$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{W} \exp(u_w^T v_c)}$$

where o is the outside (or output) word id,

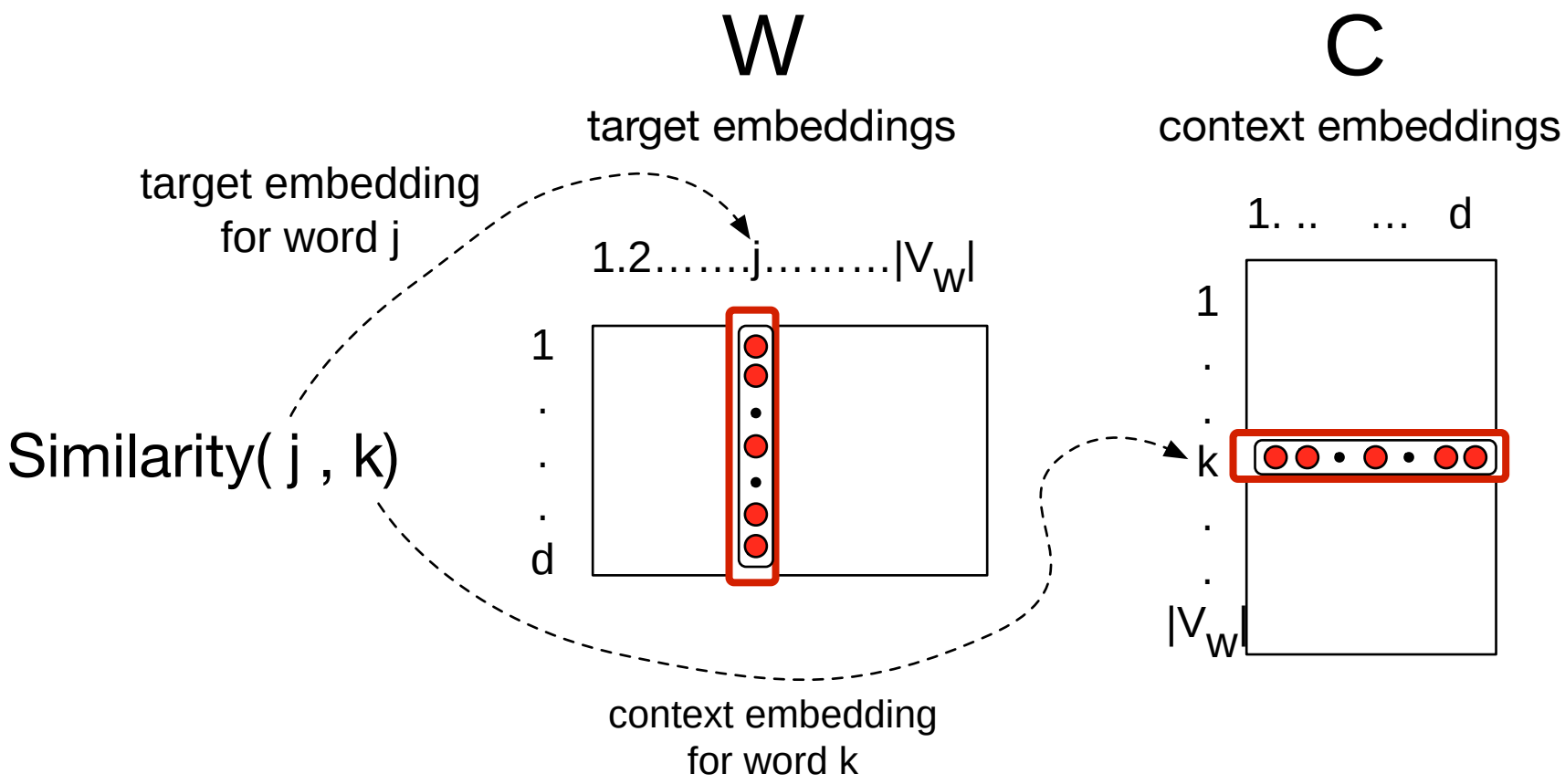
c is the center word id,

u and v are "center" and "outside" vectors of o and c

- Every word has two vectors!
- This is essentially "dynamic" logistic regression

Distributed representations of words and phrases and their compositionality [Mikolov vd.'13]

Intuition: similarity as dot-product between a target vector and context vector



- Similarity(j,k) = $c_k \cdot v_j$
- We use softmax to turn into probabilities

$$p(w_k|w_j) = \frac{exp(c_k \cdot v_j)}{\sum_{i \in |V|} exp(c_i \cdot v_j)}$$

Details of word2vec

- Predict surrounding words in a window of length m of every word.
- For $p(w_{t+j}|w_t)$ the simplest first formulation is

$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{W} \exp(u_w^T v_c)}$$

- Every word has two vectors!
- We can either:
  - Just use v_j
  - Sum them
  - Concatenate them to make a double-length embedding

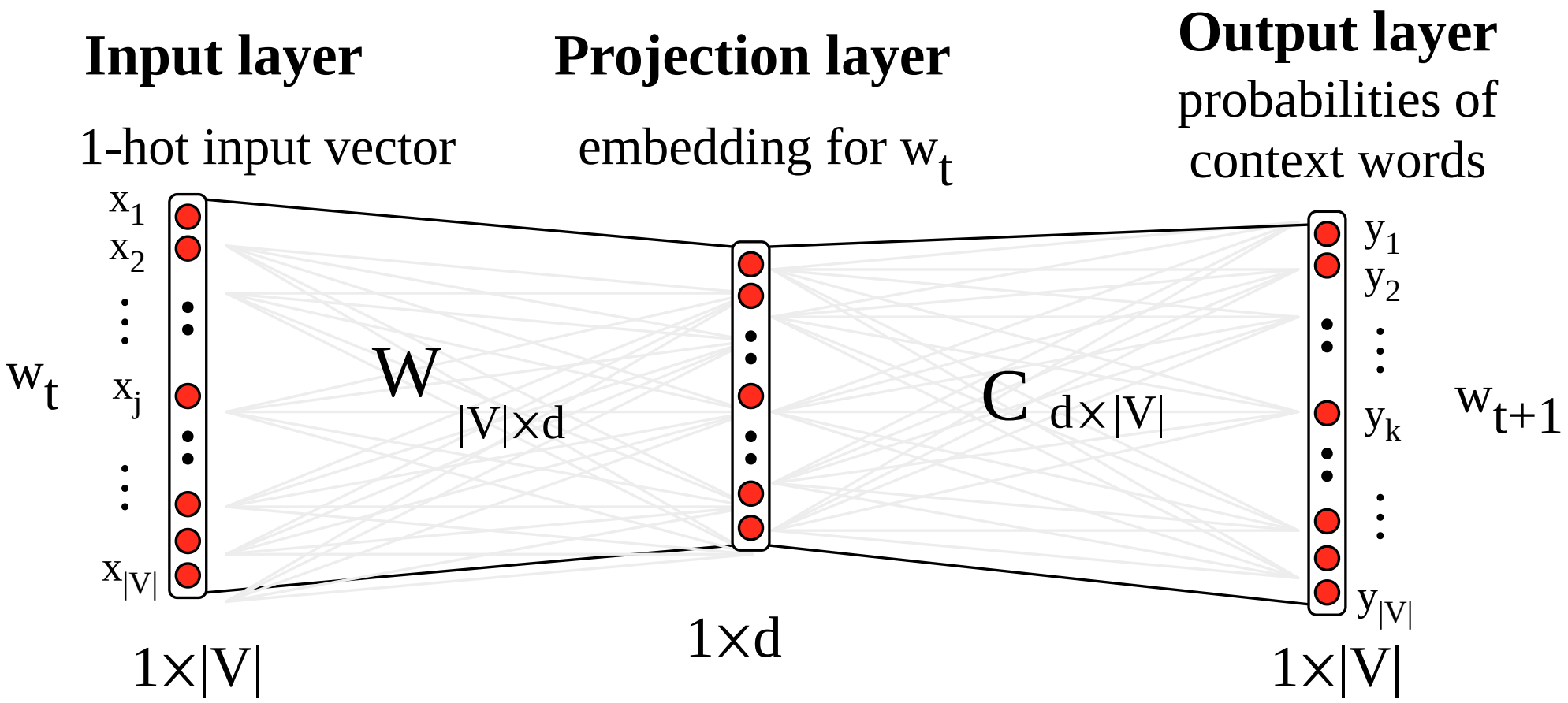
Distributed representations of words and phrases and their compositionality [Mikolov vd.'13]

Learning

- Start with some initial embeddings (e.g., random)
- iteratively make the embeddings for a word
  - more like the embeddings of its neighbors
  - less like the embeddings of other words.

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

Visualizing W and C as a network for doing error backprop



Problem with the softmax

- The denominator: have to compute over every word in vocabulary

$$p(w_k|w_j) = \frac{exp(c_k \cdot v_j)}{\sum_{i \in |V|} exp(c_i \cdot v_j)}$$

- Instead: just sample a few of those negative words

Skipgram with negative sampling: Loss function

$$\log \sigma(c \cdot w) + \sum_{i=1}^{\kappa} \mathbb{E}_{w_i \sim p(w)} \left[\log \sigma(-w_i \cdot w) \right]$$

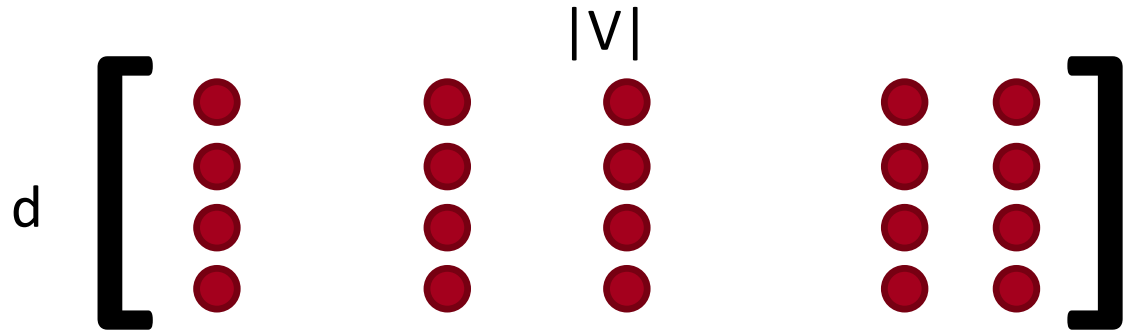
Stochastic gradients with word vectors!

- But in each window, we only have at most 2c -1 words, so $\nabla_\theta J_t(\theta)$ is very sparse!

$$\nabla_\theta J_t(\theta) = \begin{bmatrix} 0 \\ \vdots \\ \nabla_{v_{like}} \\ \vdots \\ 0 \\ \nabla_{u_I} \\ \vdots \\ \nabla_{u_{learning}} \\ \vdots \end{bmatrix} \in \mathbb{R}^{2dV}$$

Stochastic gradients with word vectors!

- We may as well only update the word vectors that actually appear!
- Solution: either keep around hash for word vectors or only update certain columns of full embedding matrix U and V



- Important if you have millions of word vectors and do distributed computing to not have to send gigantic updates around.

Embeddings capture semantics!

- Words similar to **“frog”**
  1. frogs
  2. toad
  3. litoria
  4. leptodactylidae
  5. rana
  6. lizard
  7. eleutherodactylus

“litoria”

“leptodactylidae”

“rana”

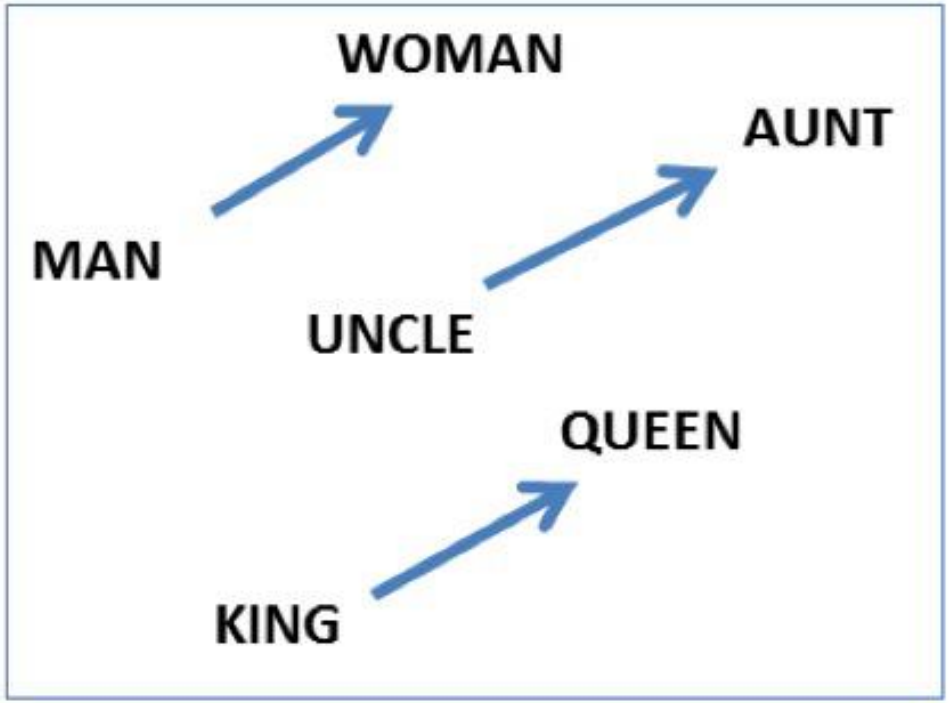
“eleutherodactylus”

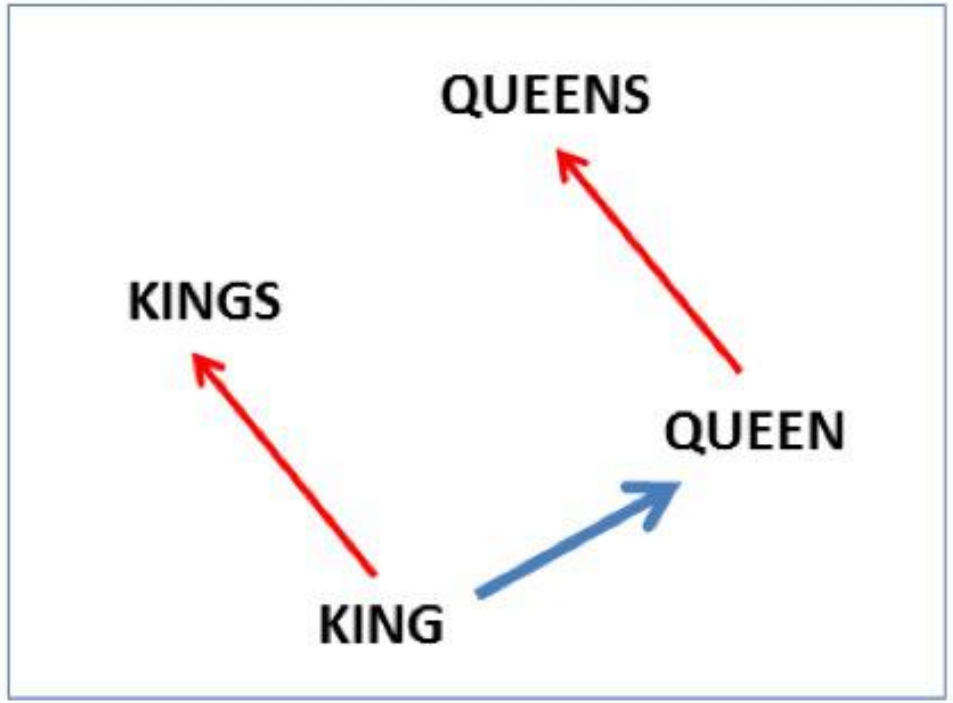
GloVe: Global Vectors for Word Representation [Pennington vd.'14]

Embeddings capture relational meaning!

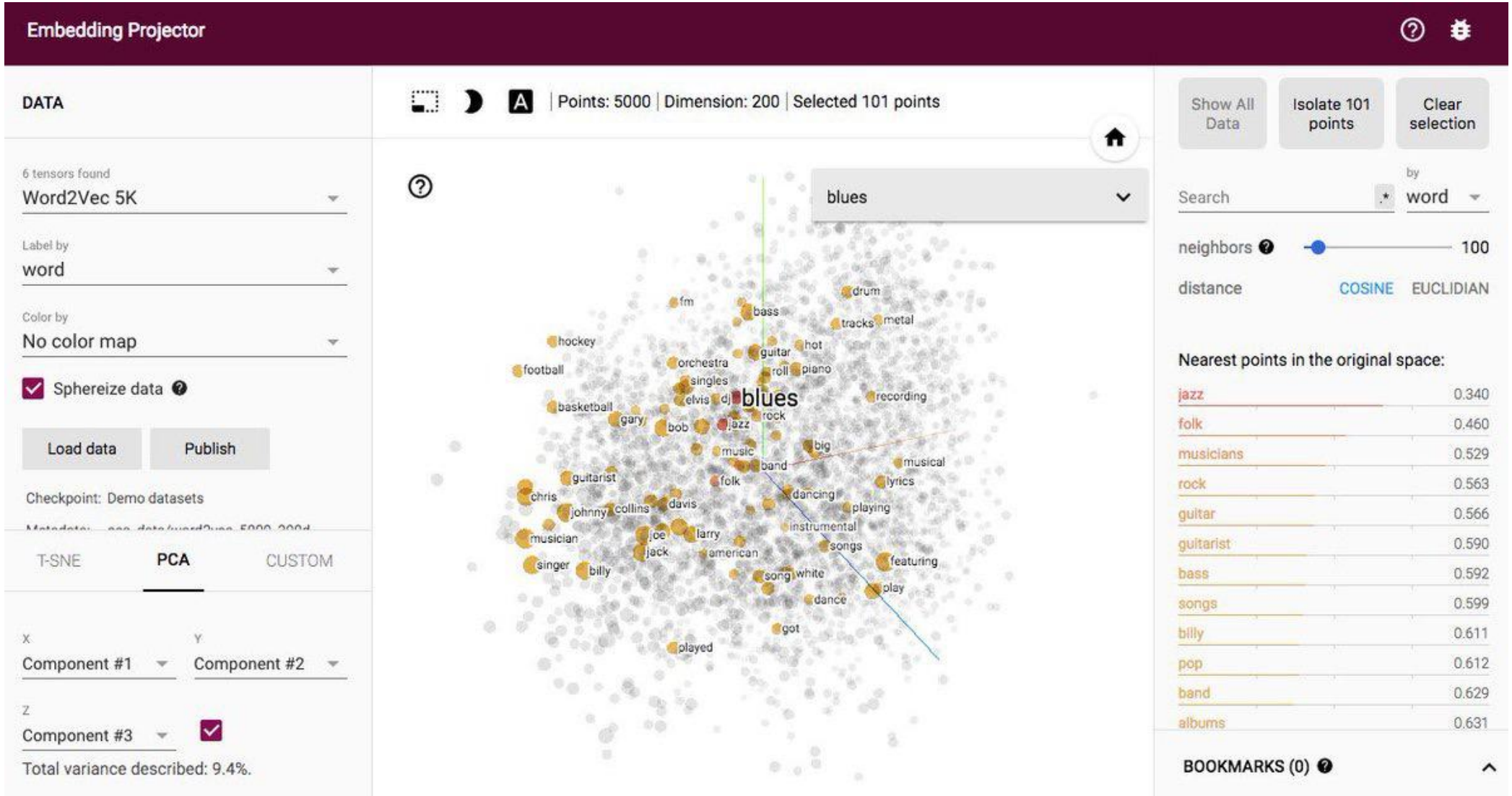
vector(*'king'*) - vector(*'man'*) + vector(*'woman'*) ≈ vector('queen')

vector(*'Paris'*) - vector(*'France'*) + vector(*'Italy'*) ≈ vector('Rome')





Demo time



http://projector.tensorflow.org

Next Lecture:

Training Deep Neural Networks