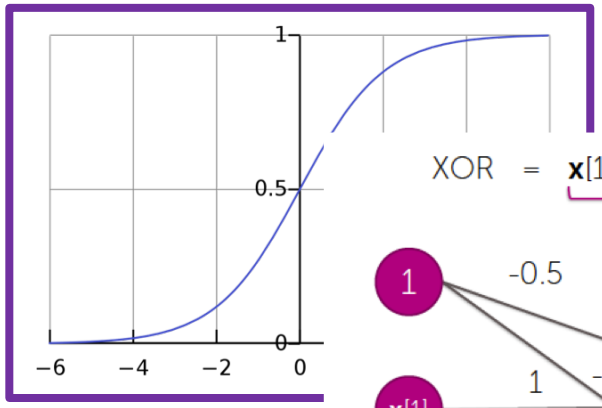
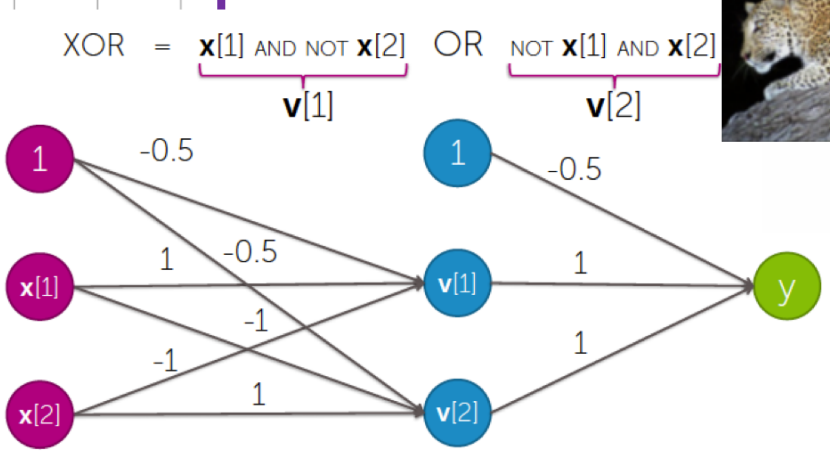
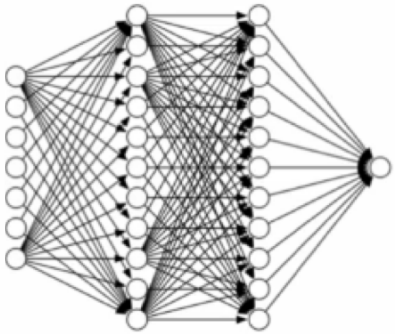
Tufts COMP 135: Introduction to Machine Learning

https://www.cs.tufts.edu/comp/135/2019s/

Neural Networks 2/2







Many slides attributable to:
Erik Sudderth (UCI), Emily Fox (UW),
Finale Doshi-Velez (Harvard)
James, Witten, Hastie, Tibshirani (ISL/ESL books)

Prof. Mike Hughes

Logistics

- Project 1: Keep going!
- Recitation tonight Monday
  - Hands-on intro to neural nets
  - With *automatic differentiation*
- HW4 out on Wed, due in TWO WEEKS

Objectives Today: Neural Networks Unit 2/2

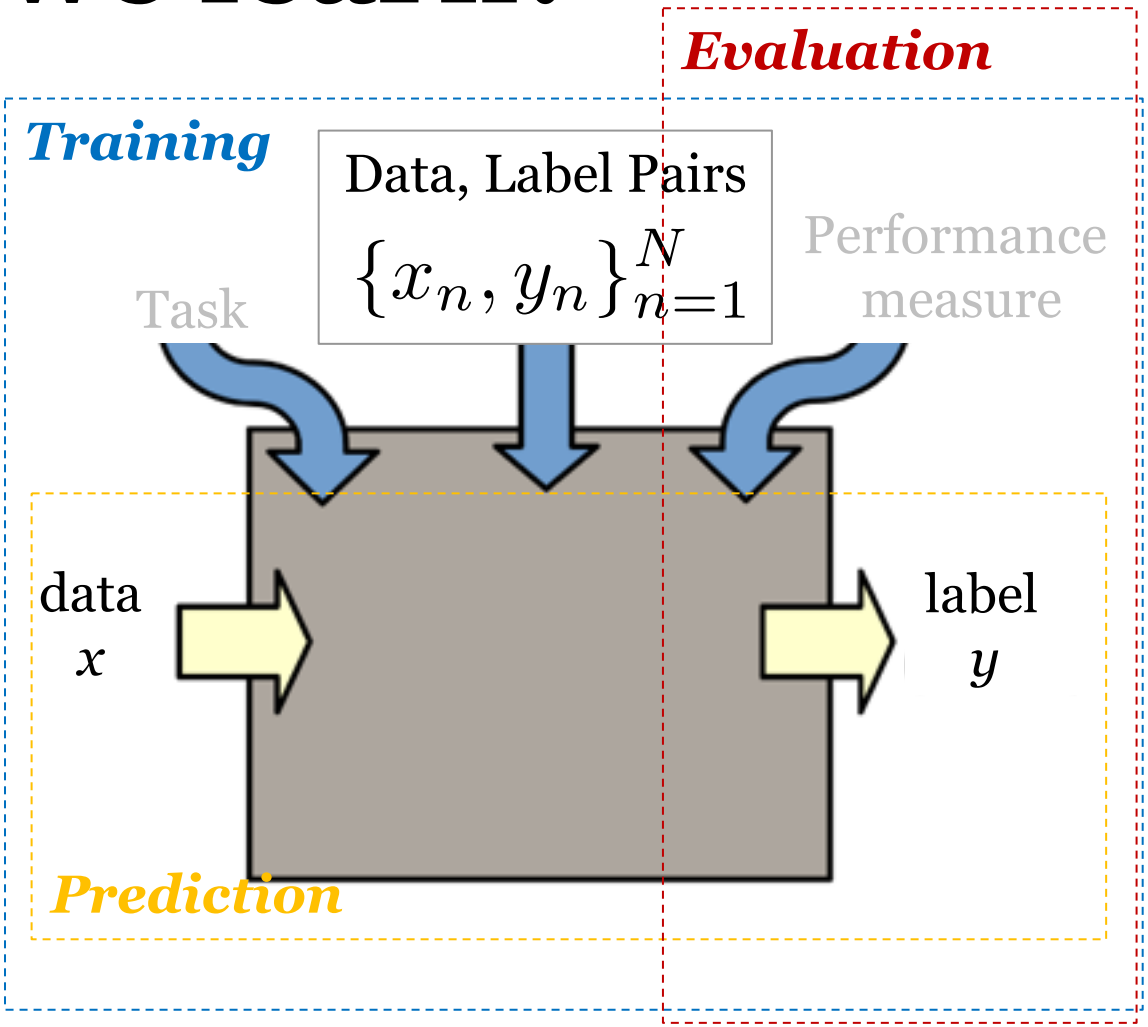
- Review: **learning** feature representations
  - Feed-forward neural nets (MLPs)
  - Activation functions
- Loss functions for multi-class classification
- Training via gradient descent
  - Back-propagation = gradient descent + chain rule
  - Automatic differentiation

What will we learn?

Supervised Learning

Unsupervised Learning

Reinforcement Learning



Task: Binary Classification

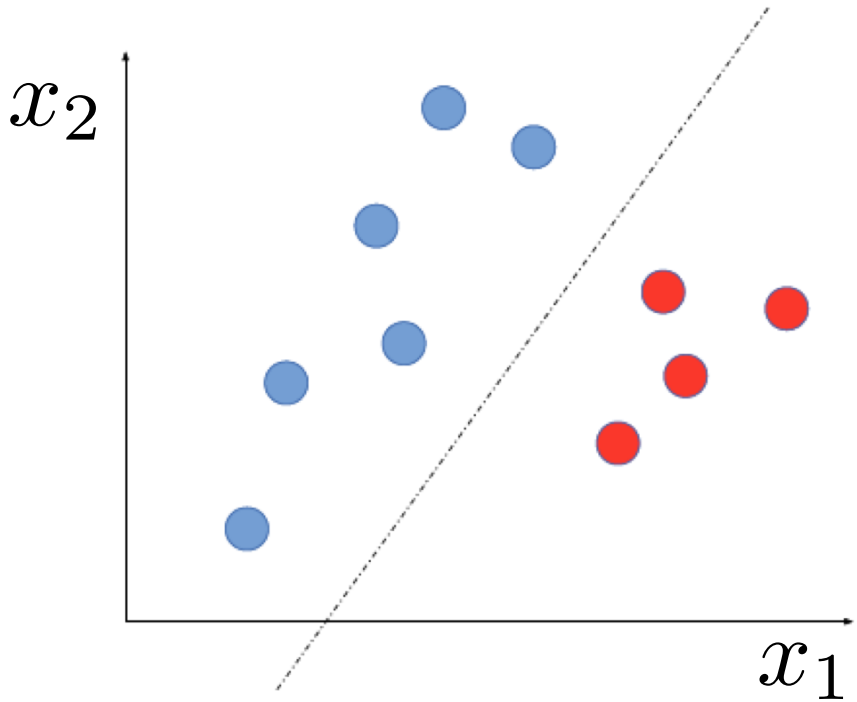
Supervised Learning

binary classification

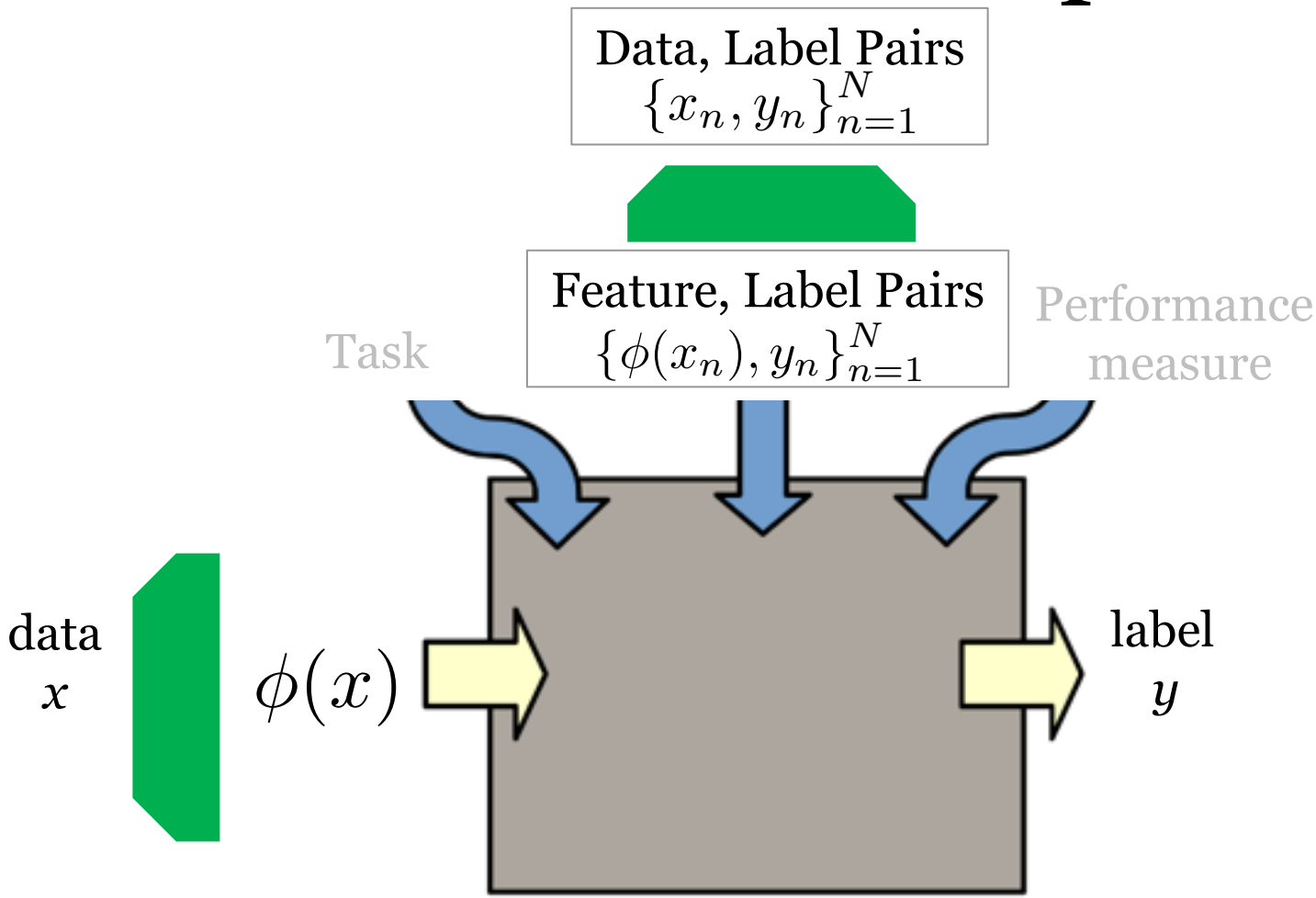
Unsupervised Learning

Reinforcement Learning

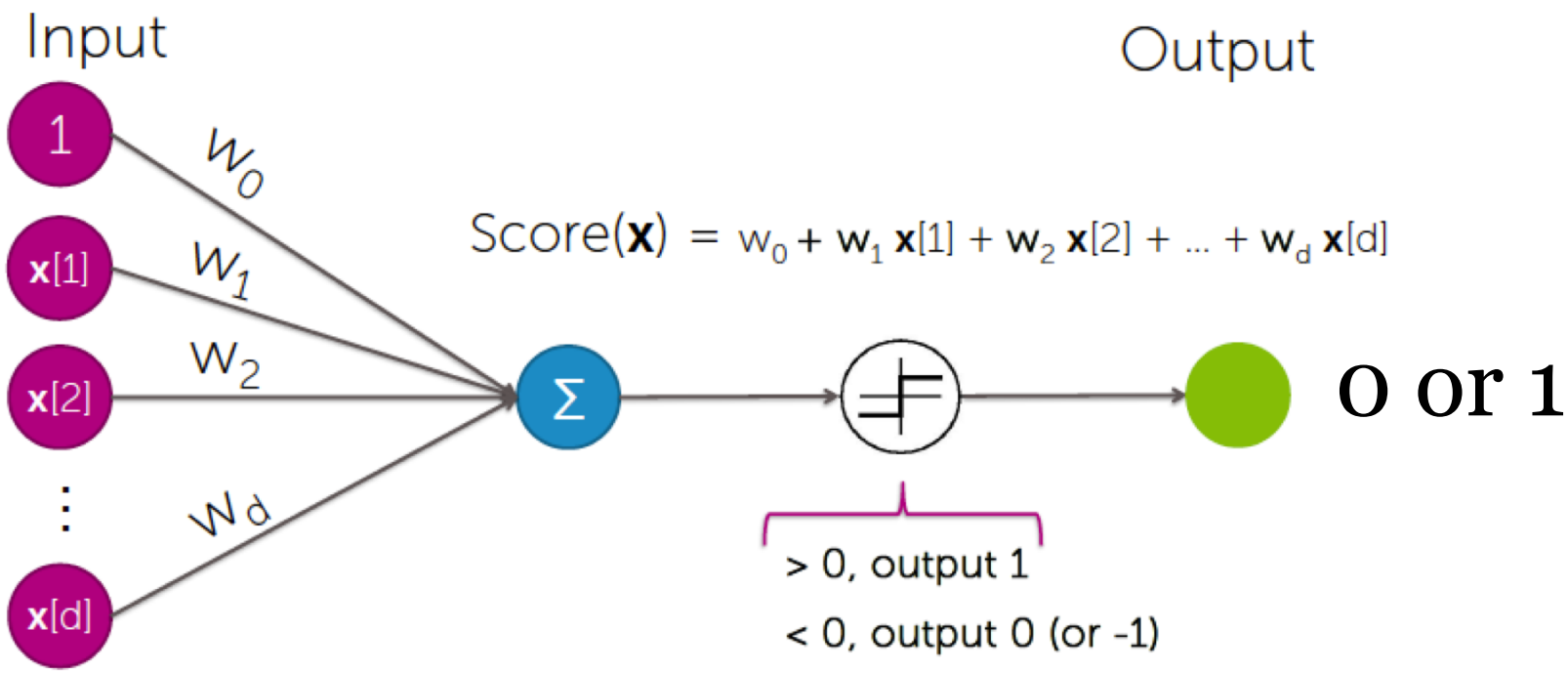
y is a binary variable (red or blue)



Feature Transform Pipeline

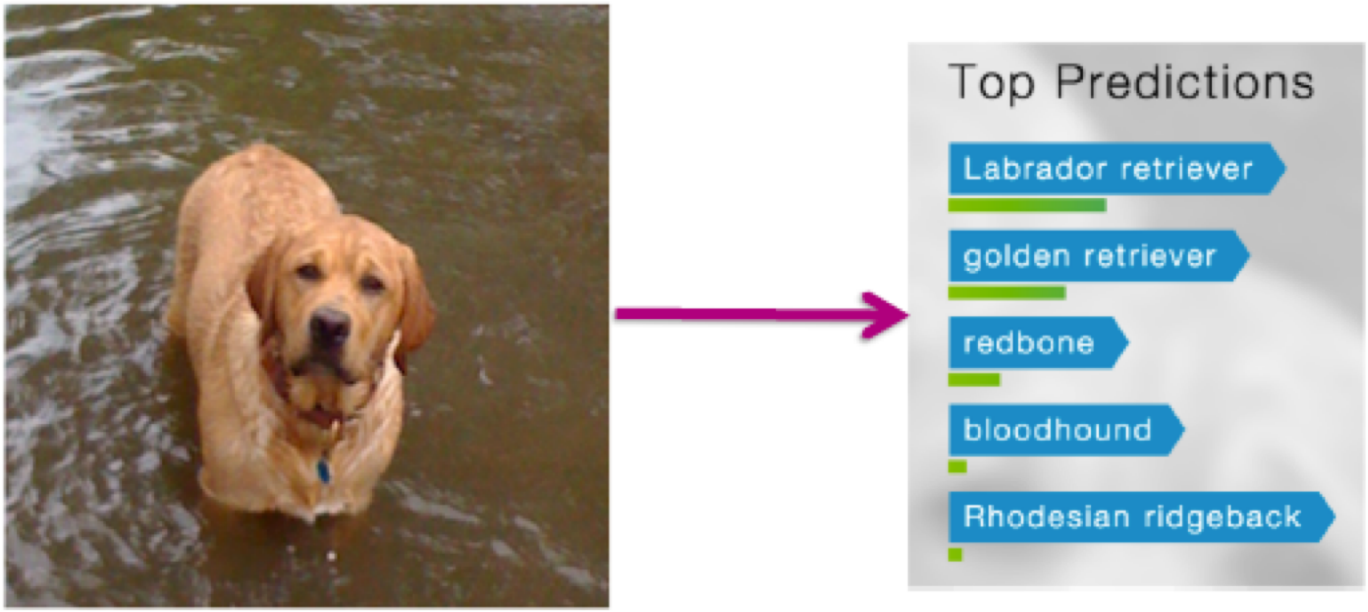


Logistic Regr. Network Diagram



Credit: Emily Fox (UW)
https://courses.cs.washington.edu/courses/cse416/18sp/slides/

Multi-class Classification



Input: x
Image pixels

Output: y
Predicted object

How to do this?

Binary Prediction

Goal: Predict label (0 or 1) given features x

- Input: $x_i \triangleq [x_{i1}, x_{i2}, \ldots x_{if} \ldots x_{iF}]$

  "features"
  "covariates"
  "attributes"

  Entries can be real-valued, or other numeric types (e.g. integer, binary)

- Output: $y_i \in \{0, 1\}$

  "responses" or "labels"

  Binary label (0 or 1)

```
>>> yhat_N = model.predict(x_NF)
>>> yhat_N[:5]
[0, 0, 1, 0, 1]
```

Binary Proba. Prediction

Goal: Predict probability of label given features

- Input: $x_i \triangleq [x_{i1}, x_{i2}, \ldots x_{if} \ldots x_{iF}]$

  "features"
  "covariates"
  "attributes"

  Entries can be real-valued, or other numeric types (e.g. integer, binary)

- Output: $\hat{p}_i \triangleq p(Y_i = 1 | x_i)$

  "probability"

  Value between 0 and 1
  e.g. 0.001, 0.513, 0.987

```
>>> yproba_N2 = model.predict_proba(x_NF)
>>> yproba1_N = model.predict_proba(x_NF)[:,1]
>>> yproba1_N[:5]
[0.143, 0.432, 0.523, 0.003, 0.994]
```

Multi-class Prediction

Goal: Predict one of C classes given features x

- Input: $x_i \triangleq [x_{i1}, x_{i2}, \ldots x_{if} \ldots x_{iF}]$

  "features"
  "covariates"
  "attributes"

  Entries can be real-valued, or other numeric types (e.g. integer, binary)

- Output: $y_i \in \{0, 1, 2, \ldots C-1\}$

  "responses" or "labels"

  Integer label (0 or 1 or ... or C-1)

```
>>> yhat_N = model.predict(x_NF)
>>> yhat_N[:6]
[0, 3, 1, 0, 0, 2]
```

Multi-class Proba. Prediction

Goal: Predict probability of label given features

Input: $x_i \triangleq [x_{i1}, x_{i2}, \ldots x_{if} \ldots x_{iF}]$

"features"
"covariates"
"attributes"

Entries can be real-valued, or other numeric types (e.g. integer, binary)

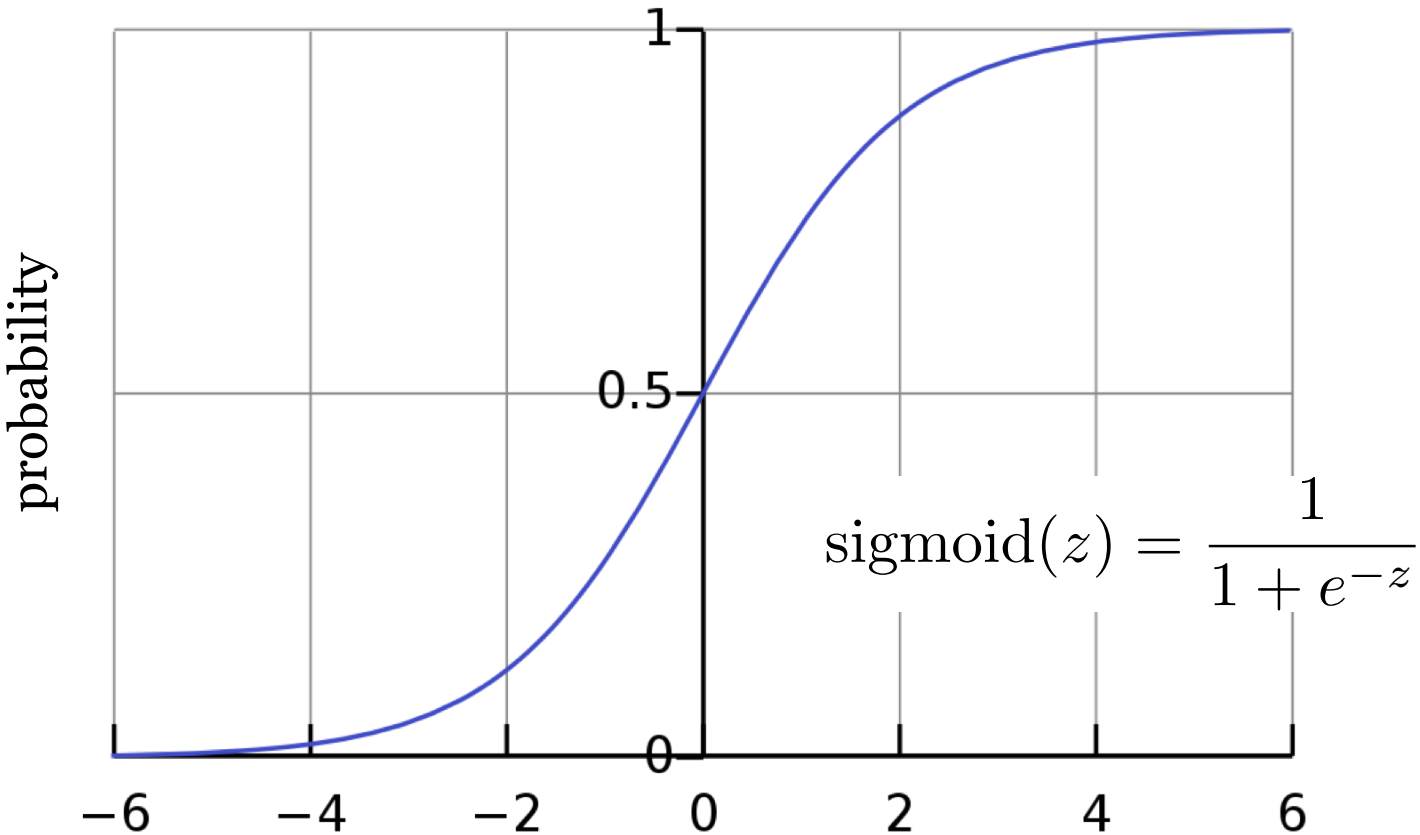
Output:

$$\hat{p}_i \triangleq [p(Y_i = 0|x_i) \quad p(Y_i = 1|x_i) \ldots \quad p(Y_i = C-1|x_i)]$$

"probability" Vector of C non-negative values, sums to one

```
>>> yproba_NC = model.predict_proba(x_NF)
>>> yproba_c_N = model.predict_proba(x_NF)[:,c]
>>> np.sum(yproba_NC, axis=1)
[1.0, 1.0, 1.0, 1.0]
```

From Real Value to Probability



From Vector of Reals to Vector of Probabilities

$$z_i = \begin{bmatrix} z_{i1} & z_{i2} & \ldots & z_{ic} & \ldots & z_{iC} \end{bmatrix}$$

$$\hat{p}_i = \begin{bmatrix} \frac{e^{z_{i1}}}{\sum_{c=1}^{C} e^{z_{ic}}} & \frac{e^{z_{i2}}}{\sum_{c=1}^{C} e^{z_{ic}}} & \ldots & \ldots & \frac{e^{z_{iC}}}{\sum_{c=1}^{C} e^{z_{ic}}} \end{bmatrix}$$

called the "softmax" function

Representing multi-class labels

$$y_n \in \{0, 1, 2, \ldots C-1\}$$

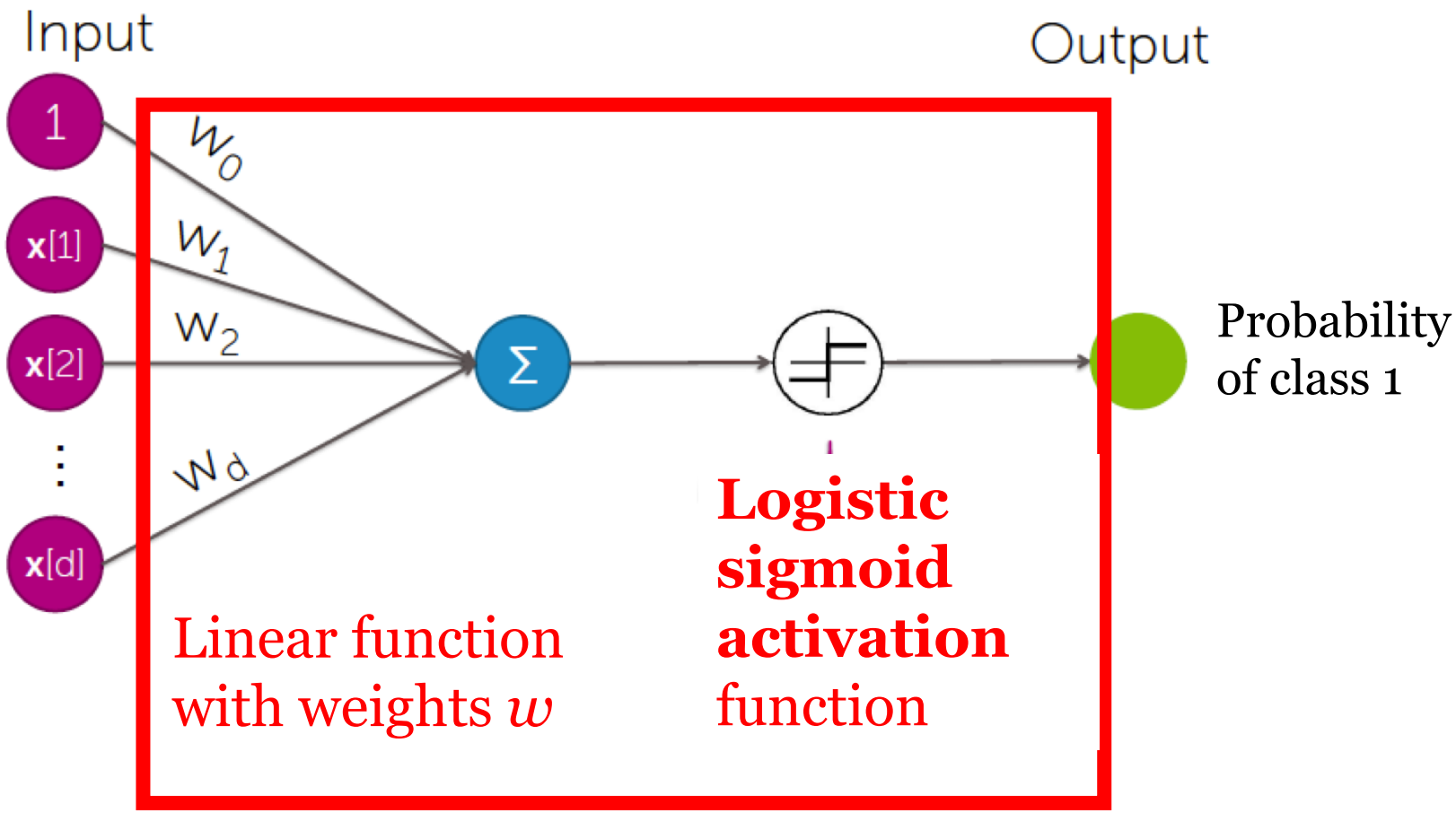
Encode as length-C ***one hot binary*** vector

$$\bar{y}_n = [\bar{y}_{n1} \;\; \bar{y}_{n2} \;\; \ldots \;\; \bar{y}_{nc} \;\; \ldots \;\; \bar{y}_{nC}]$$

```
Examples (assume C=4 labels)

    class 0:   [1 0 0 0]
    class 1:   [0 1 0 0]
    class 2:   [0 0 1 0]
    class 3:   [0 0 0 1]
```

"Neuron" for Binary Prediction



Credit: Emily Fox (UW)

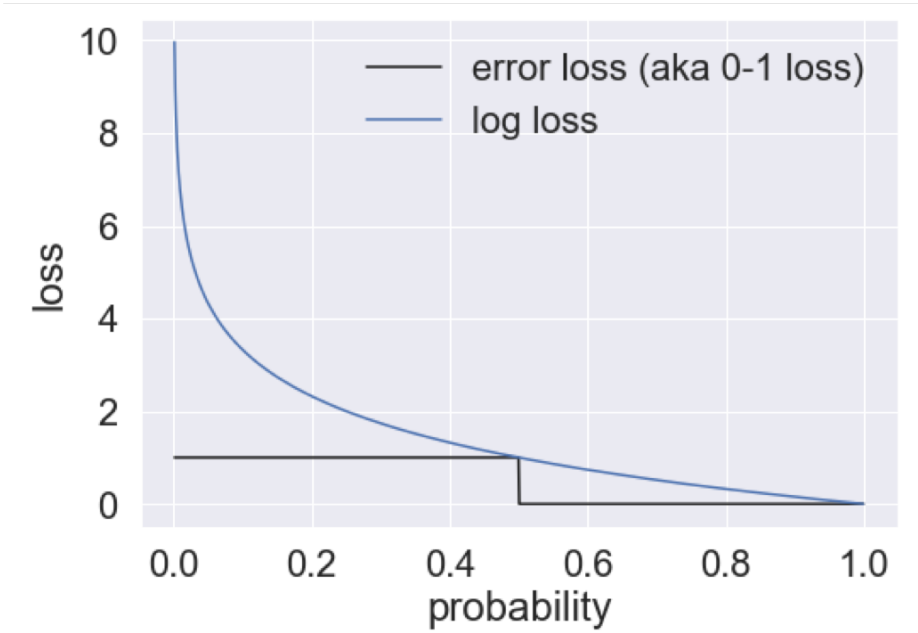
Neurons for Multi-class Prediction

- Can you draw it?

Recall: Binary log loss

$$\text{error}(y, \hat{y}) = \begin{cases} 1 & \text{if } y \neq \hat{y} \\ 0 & \text{if } y = \hat{y} \end{cases}$$

$$\text{log\_loss}(y, \hat{p}) = -y \log \hat{p} - (1 - y) \log(1 - \hat{p})$$



Plot assumes:

- True label is 1

- Threshold is 0.5

- Log base 2

Multi-class log loss

Input: two vectors of length C
Output: scalar value (strictly non-negative)

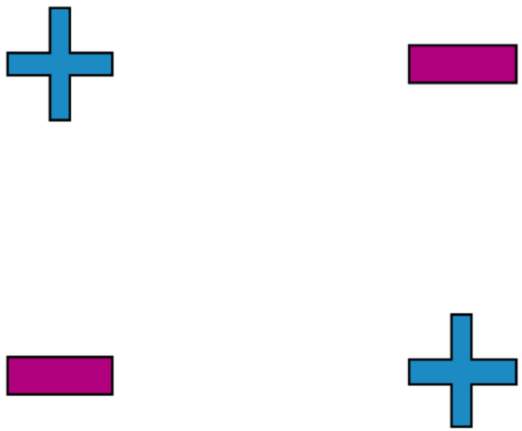
$$\text{log\_loss}(\bar{y}_n, \hat{p}_n) = -\sum_{c=1}^{C} \bar{y}_{nc} \log \hat{p}_{nc}$$

Justifications carry over from the binary case:
- Interpret as upper bound on the error rate
- Interpret as cross entropy of multi-class discrete random variable
- Interpret as log likelihood of multi-class discrete random variable

MLP: Multi-Layer Perceptron

1 or more hidden layers
followed by 1 output layer

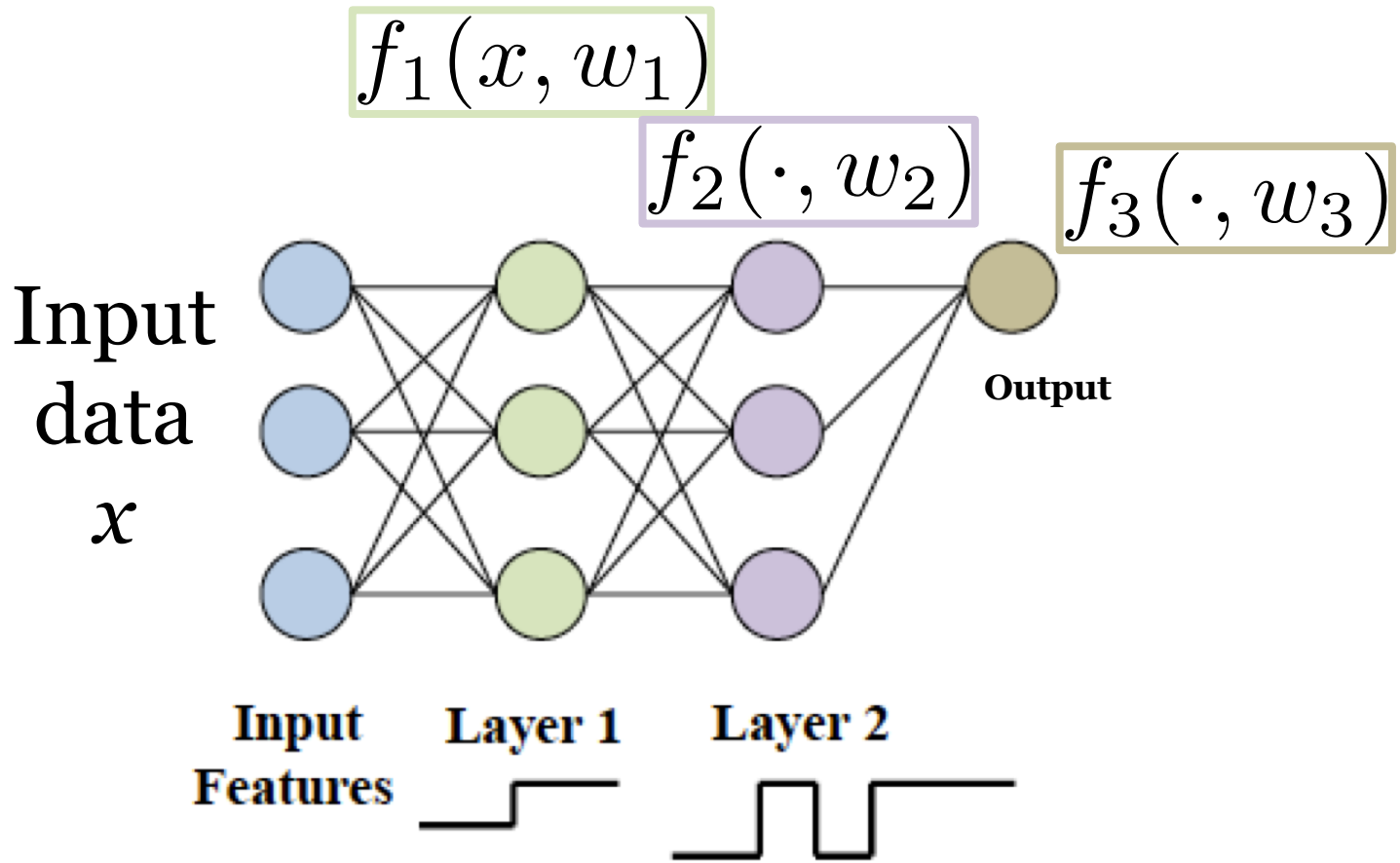
Linear decision boundaries cannot solve XOR



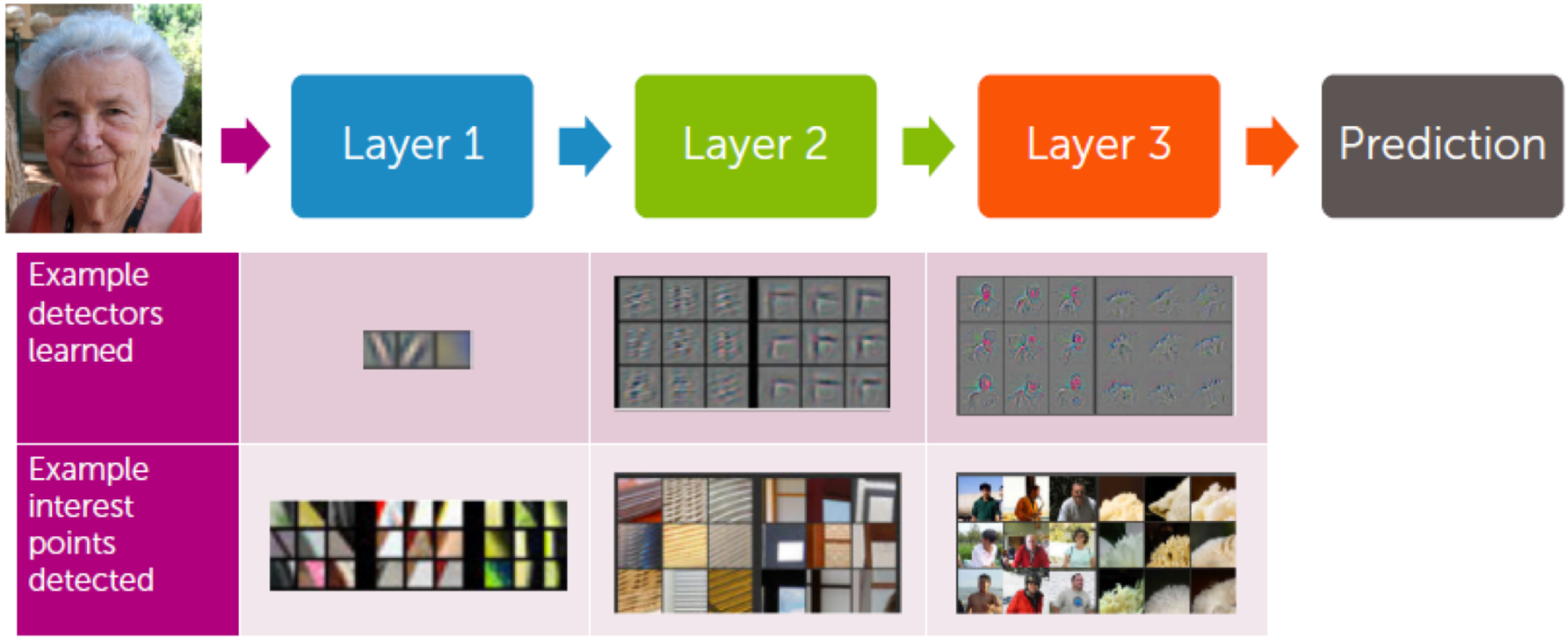
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

XOR = **x**[1] AND NOT **x**[2] OR NOT **x**[1] AND **x**[2]

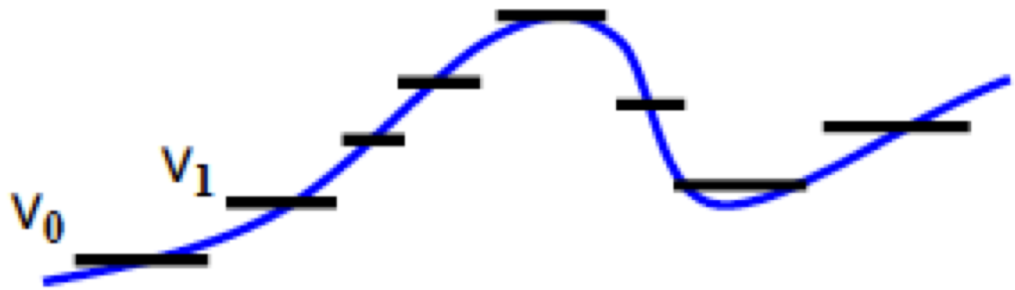
Diagram of an MLP



Each Layer Extracts "Higher Level" Features



MLPs with 1 hidden layer can approximate any functions with enough hidden units!



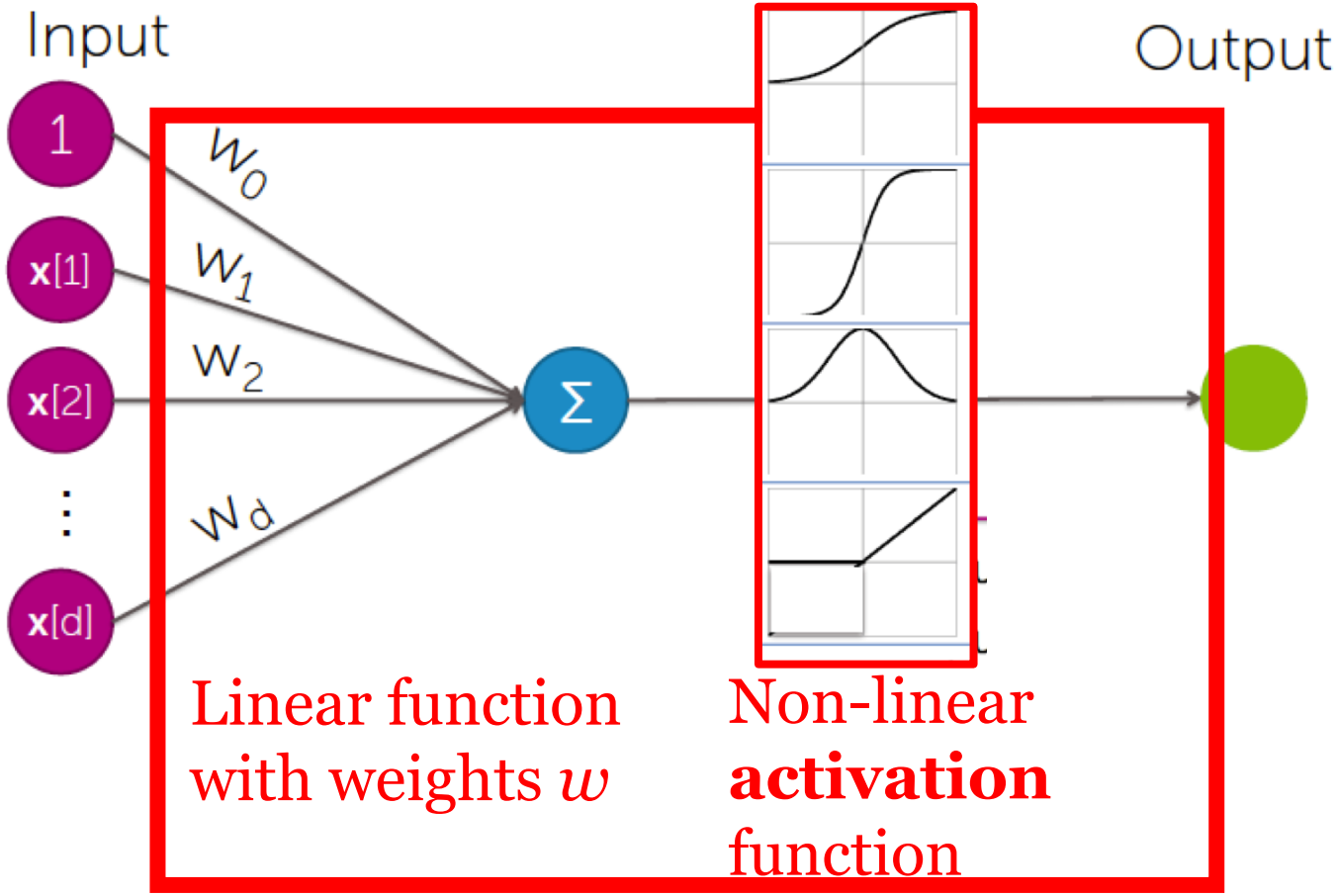
Universal approximation theorem

From Wikipedia, the free encyclopedia

In the mathematical theory of artificial neural networks, the **universal approximation theorem** states[1] that a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbf{R}^n$, under mild assumptions on the activation function. The theorem thus states that simple neural networks can *represent* a wide variety of interesting functions when given appropriate parameters; however, it does not touch upon the algorithmic learnability of those parameters.

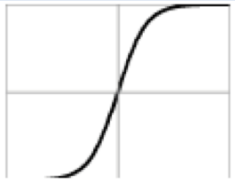
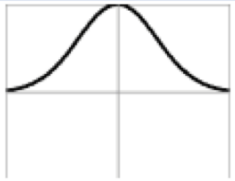
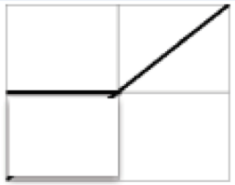
One of the first versions of the theorem was proved by George Cybenko in 1989 for sigmoid activation functions.[2]

Which Activation Function?



Credit: Emily Fox (UW)

Activation Functions

Logistic	$\sigma(z) = \frac{1}{1+\exp(-z)}$		$\frac{\partial \sigma}{\partial z}(z) = \sigma(z)(1-\sigma(z))$
Hyperbolic Tangent	$\sigma(z) = \frac{1-\exp(-2z)}{1+\exp(-2z)}$		$\frac{\partial \sigma}{\partial z}(z) = 1-(\sigma(z))^2$
Gaussian	$\sigma(z) = \exp(-z^2/2)$		$\frac{\partial \sigma}{\partial z}(z) = -z\sigma(z)$
ReLU (rectified linear)	$\sigma(z) = \max(0, z)$		$\frac{\partial \sigma}{\partial z}(z) = \mathbb{1}[z > 0]$
Linear	$\sigma(z) = z$		and many others...

How to train Neural Nets?

Just like logistic regression

Set up a loss function

Apply Gradient Descent!

Review: LR notation

- Feature vector with first entry constant

$$\tilde{x}_n = [1 \; x_{n1} \; x_{n2} \; \ldots \; x_{nF}]$$

- Weight vector (first entry is the "bias")

$$w = [w_0 \quad w_1 \quad w_2 \ldots w_F]$$

- "Score" value z (real number, -inf to +inf)

$$z_n = w^T \tilde{x}_n$$

Review: Gradient of LR

$$\boxed{z_n = w^T \tilde{x}_n}$$

Log likelihood

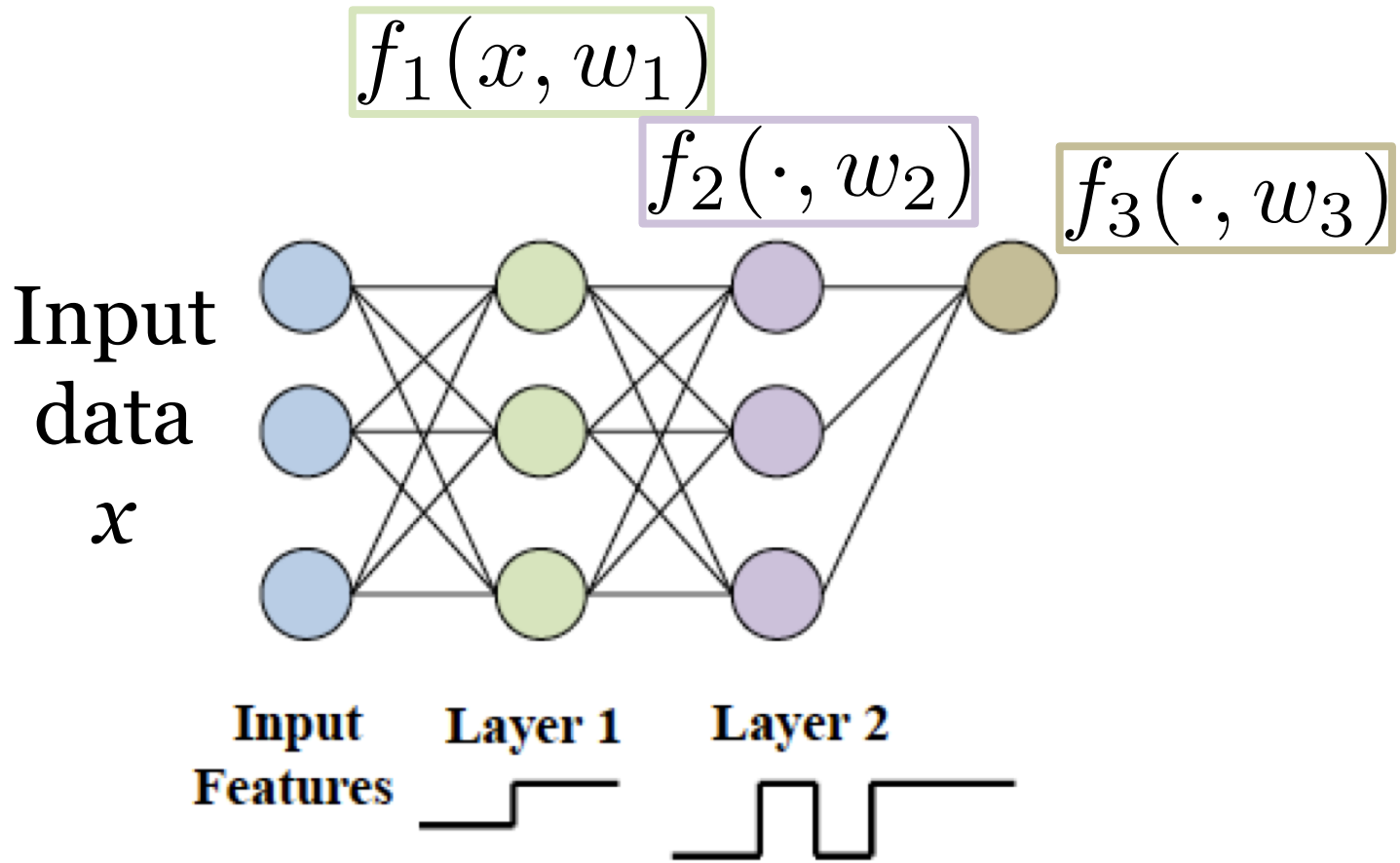
$$J(z_n(w)) = y_n z_n - \log(1 + e^{z_n})$$

Gradient w.r.t. weight on feature f

$$\frac{d}{dw_f} J(z_n(w)) = \frac{d}{dz_n} J(z_n) \cdot \frac{d}{dw_f} z(w)$$

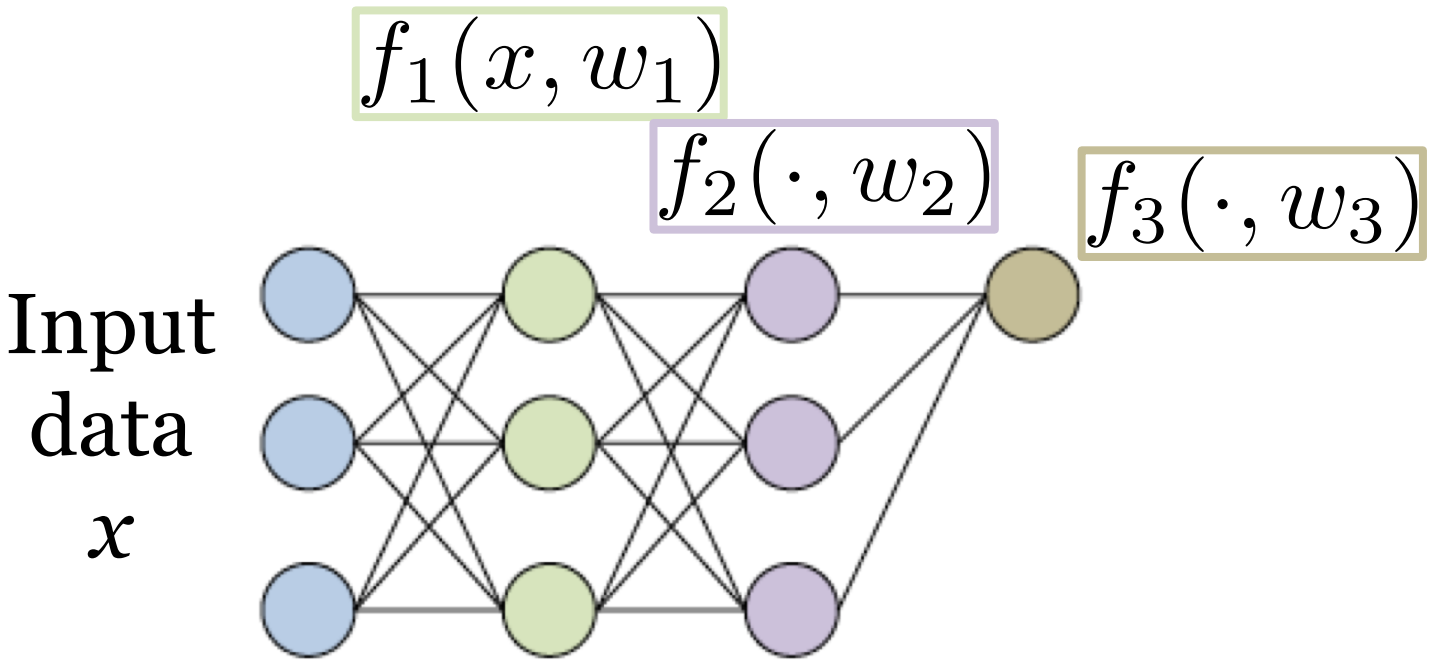
chain rule!

MLP : Composable Functions



Output as function of x

$$f_3(f_2(f_1(x, w_1), w_2), w_3)$$



Minimizing loss for composable functions

$$\min_{w_1, w_2, w_3} \sum_{n=1}^{N} \text{loss}(y_n, f_3(f_2(f_1(x_n, w_1), w_2), w_3)$$

Loss can be:

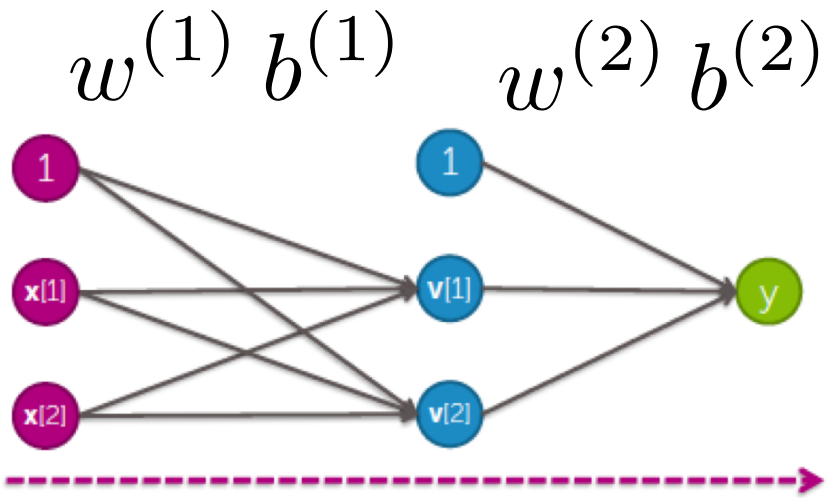
- Squared error for regression problems
- Log loss for multi-way classification problems
- … many others possible!

Compute loss via Forward Propagation

For fixed weights, forming predictions is easy!

Compute values left to right

1. Inputs: **x**[1],...,**x**[d]
2. Hidden: **v**[1],...,**v**[d]
3. Output: y

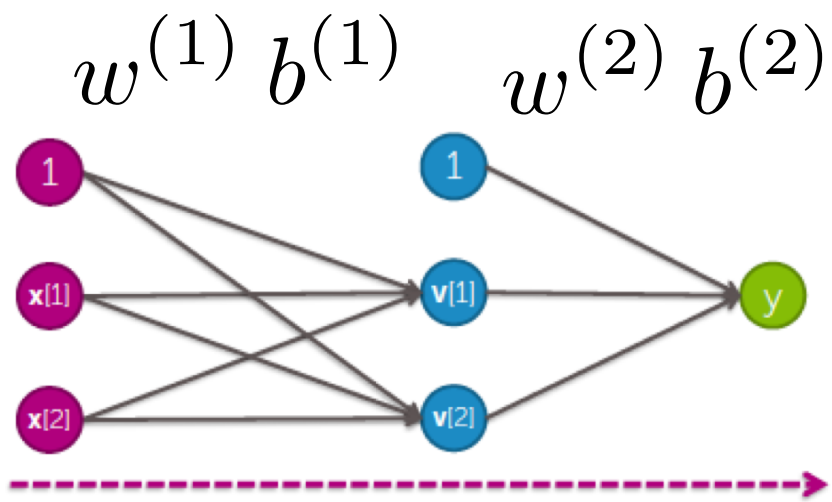


Compute loss via Forward Propagation

For fixed weights, forming predictions is easy!

Compute values left to right

1. Inputs: **x**[1],...,**x**[d]
2. Hidden: **v**[1],...,**v**[d]
3. Output: y



Step 2:

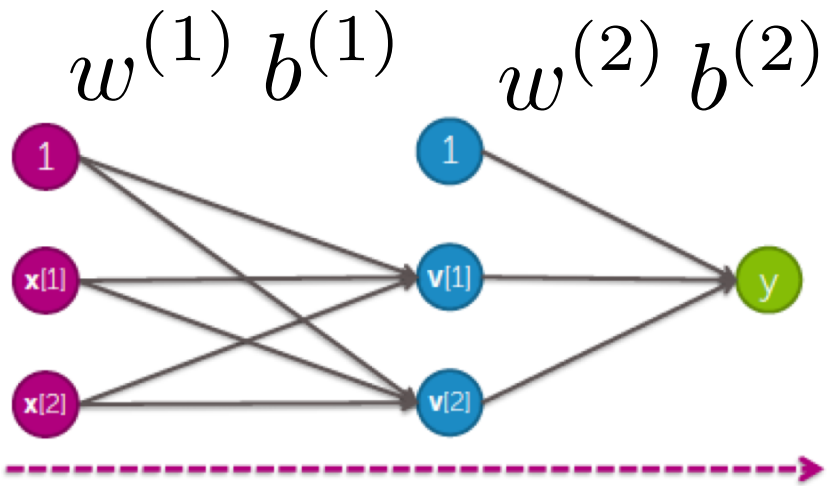
```
v = activation(np.dot(w1, x) + b1)
```

Compute loss via Forward Propagation

For fixed weights, forming predictions is easy!

Compute values left to right

1. Inputs: **x**[1],...,**x**[d]
2. Hidden: **v**[1],...,**v**[d]
3. Output: y



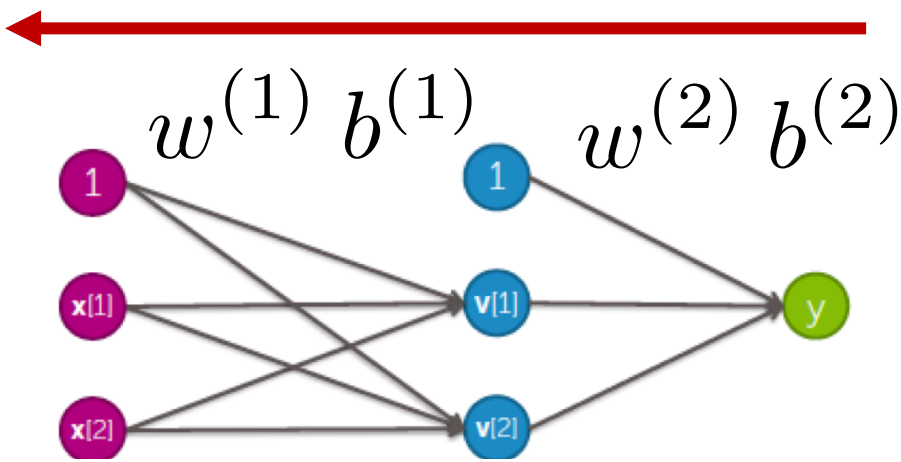
Step 2:

```
v = activation(np.dot(w1, x) + b1)
```

Step 3:

```
yhat = np.dot(w2, v) + b2
```

Compute gradient via **Back Propagation**



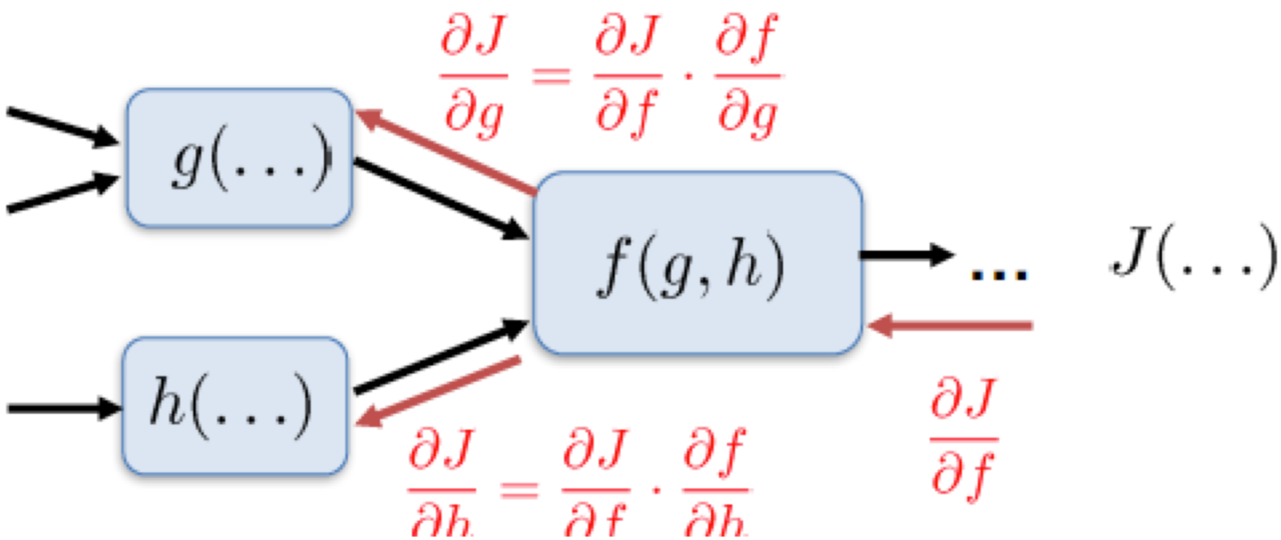
Visual Demo:

https://google-developers.appspot.com/machine-learning/crash-course/backprop-scroll/

Network view of Backprop

Think of NNs as "schematics" made of smaller functions

- Building blocks: summations & nonlinearities
- For derivatives, just apply the chain rule, etc!



H

Is Symbolic Differentiation the only way to compute derivatives?

$$f = \exp\left(\exp(x) + \exp(x)^2\right) + \sin\left(\exp(x) + \exp(x)^2\right)$$

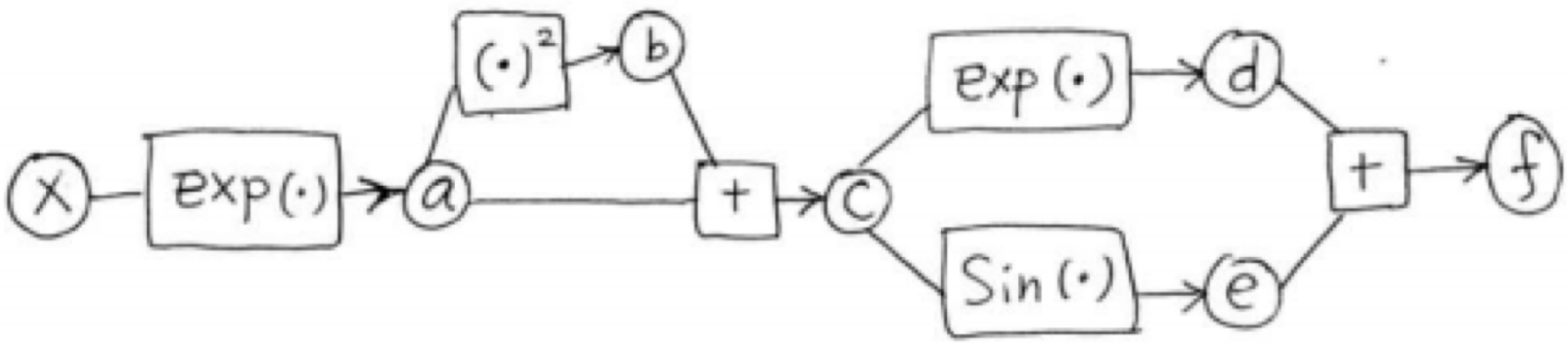
$$\frac{df}{dx} = \exp(\exp(x) + \exp(x)^2)(\exp(x) + 2\exp(x)^2) + \cos(\exp(x) + \exp(x)^2)(\exp(x) + 2\exp(x)^2)$$

Credit: Justin Domke (UMass)
https://people.cs.umass.edu/~domke/courses/sml/09autodiff_nnets.pdf

Another view: Computation graph

Credit: Justin Domke (UMass)
https://people.cs.umass.edu/~domke/courses/sml/09autodiff_nnets.pdf

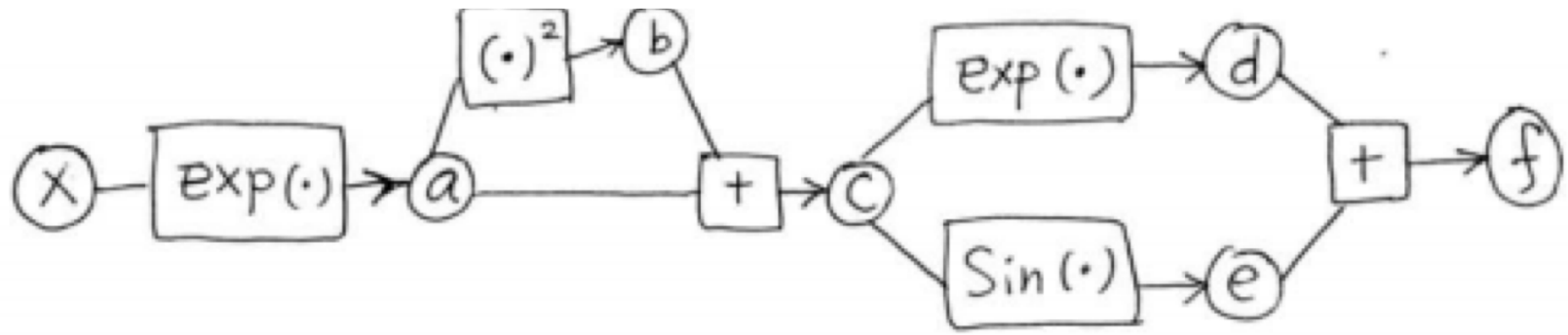
$$f = \exp\big(\exp(x) + \exp(x)^2\big) + \sin\big(\exp(x) + \exp(x)^2\big)$$



Forward Propagation

Credit: Justin Domke (UMass)
https://people.cs.umass.edu/~domke/courses/sml/09autodiff_nnets.pdf

$$f = \exp\big(\exp(x) + \exp(x)^2\big) + \sin\big(\exp(x) + \exp(x)^2\big)$$



Forward Propagation

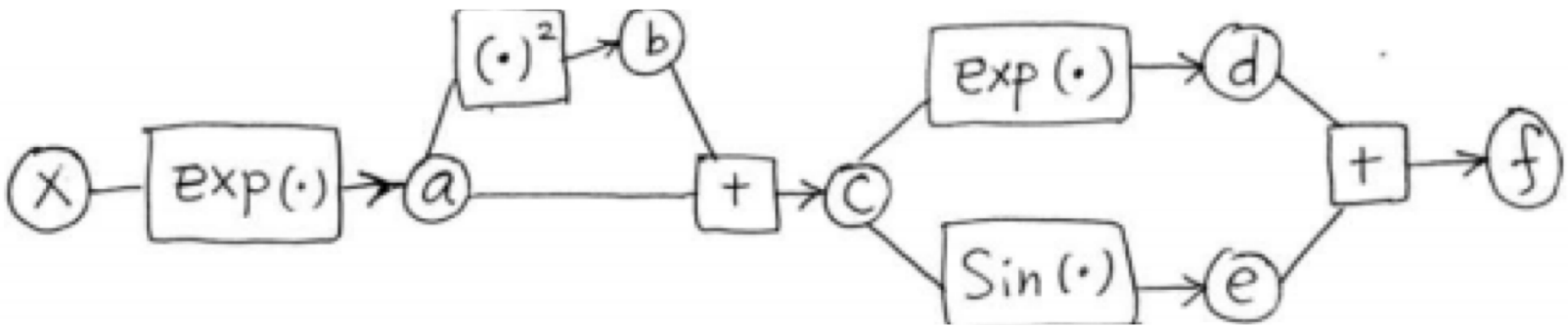
For $i = n+1, n+2 \ldots N$:

$$x_i \leftarrow g_i(\mathbf{x}_{\mathrm{Pa}(i)})$$

Back propagation

Credit: Justin Domke (UMass)
https://people.cs.umass.edu/~domke/courses/sml/09autodiff_nnets.pdf

$$f = \exp\big(\exp(x) + \exp(x)^2\big) + \sin\big(\exp(x) + \exp(x)^2\big)$$



Back Propagation

(Do forward propagation)

$$\frac{df}{dx_N} \leftarrow 1$$

For $i = N-1, N-2, \ldots 1$:

$$\frac{df}{dx_i} \leftarrow \sum_{j:i \in \mathrm{Pa}(j)} \frac{df}{dx_j}\frac{dg_j}{dx_i}.$$