Neural Networks and Autodifferentiation

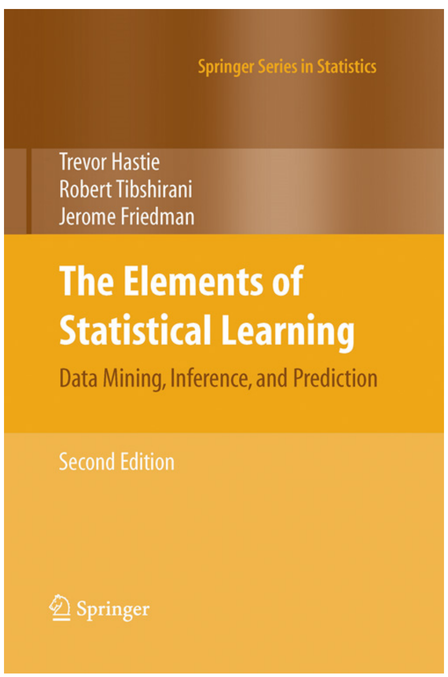
CMSC 678

UMBC

Recap from last time…

Maximum Entropy (Log-linear) Models

$$p(y \mid x) \propto \exp(\theta^T f(x, y))$$



"model the posterior probabilities of the K classes via linear functions in θ, while at the same time ensuring that they sum to one and remain in [0, 1]" ~ *Ch 4.4*

*"[The log-linear estimate] is the least biased estimate possible on the given information; i.e., it is **maximally noncommittal with regard to missing information**."* Jaynes, 1957

Normalization for Classification

$$Z = \sum_{\text{label } y} \exp\left(\begin{array}{l} \text{weight}_1 * f_1(\text{fatally shot}, y) + \\ \text{weight}_2 * f_2(\text{seriously wounded}, y) + \\ \text{weight}_3 * f_3(\text{Shining Path}, y) + \\ \dots \end{array} \right)$$

Connections to Other Techniques

Log-Linear Models

(Multinomial) logistic regression
Softmax regression
Max`imum Entropy models (MaxEnt)
Generalized Linear Models
Discriminative Naïve Bayes
Very shallow (sigmoidal) neural nets

$$y = \sum_k \theta_k x_k + b$$

the *response* can be a general (transformed) version of another response

logistic regression $$\frac{\log p(x = i)}{\log p(x = K)} = \sum_k \theta_k f(x_k, i) + b$$

Log-Likelihood Gradient

Each component *k* is the difference between:

the total value of feature f_k in the training data

$$\sum_i f_k(x_i, y_i)$$

and

the total value the current model p_θ *thinks* it computes for feature f_k

$$\sum_i \mathbb{E}_{y' \sim p}[f(x_i, y')]$$

Outline

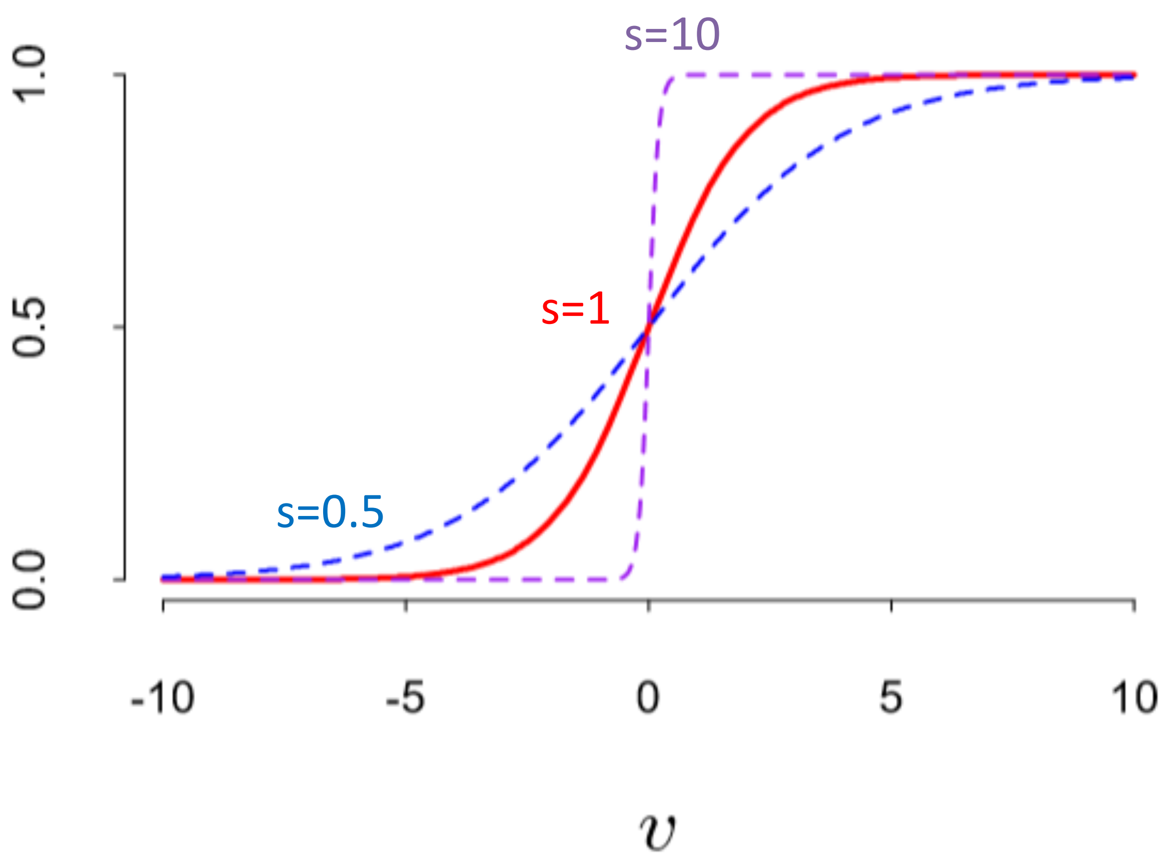
Neural networks: non-linear classifiers

Learning weights: backpropagation of error

Autodifferentiation (in reverse mode)

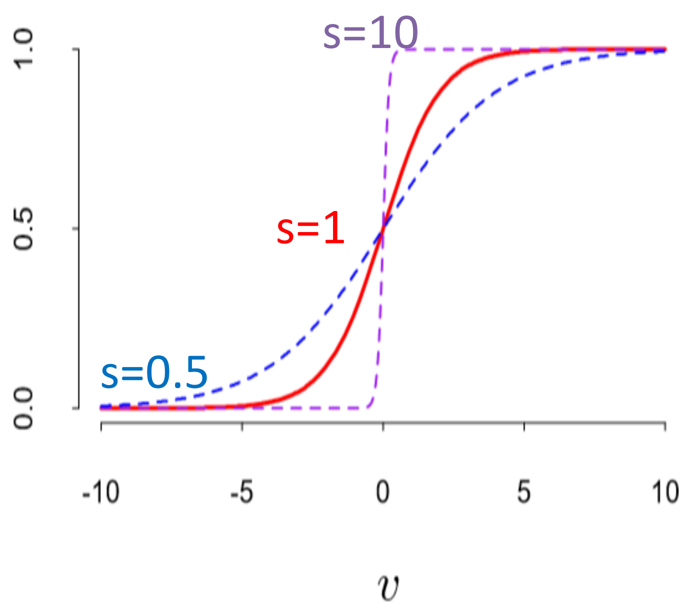
Sigmoid

$$\sigma(v) = \frac{1}{1 + \exp(-sv)}$$



Sigmoid

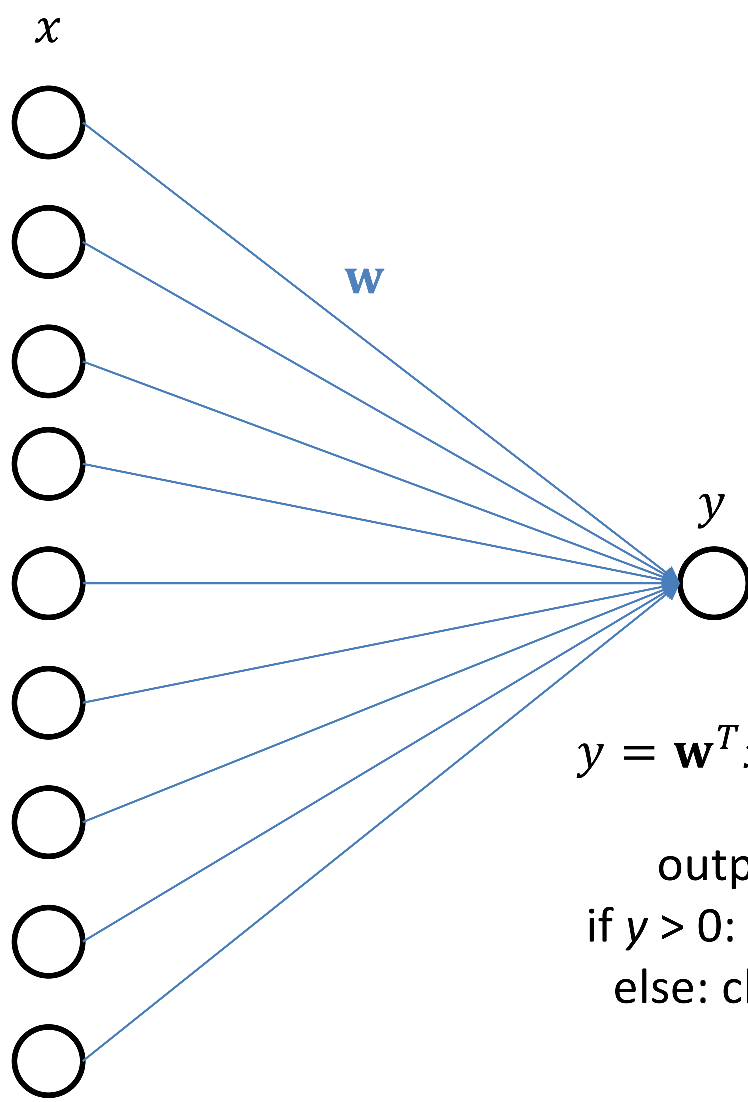
$$\sigma(v) = \frac{1}{1 + \exp(-sv)}$$



$$\frac{\partial \sigma(v)}{\partial v} = s * \sigma(v) * \big(1 - \sigma(v)\big)$$

calc practice: verify for yourself

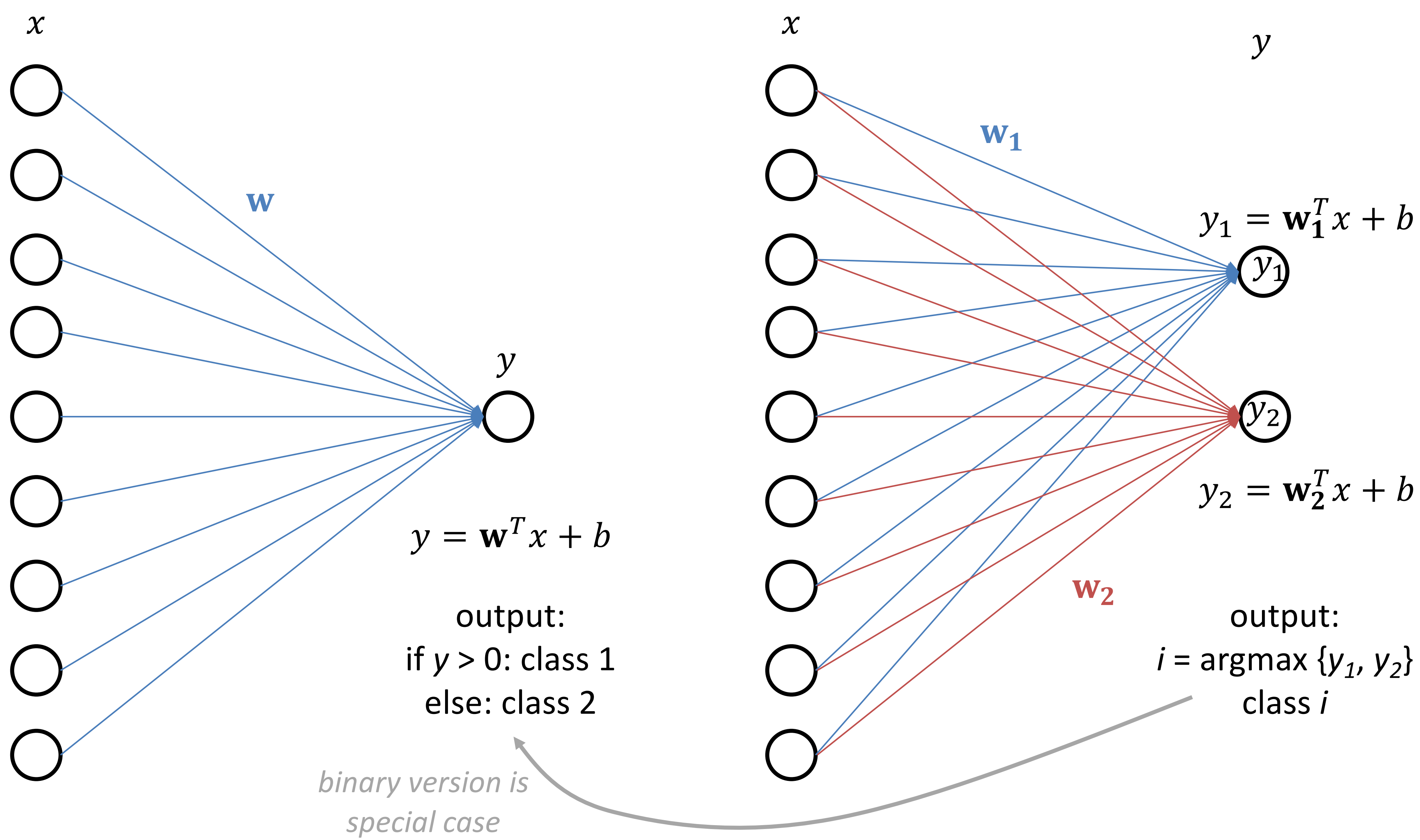
Remember Multi-class Linear Regression/Perceptron?



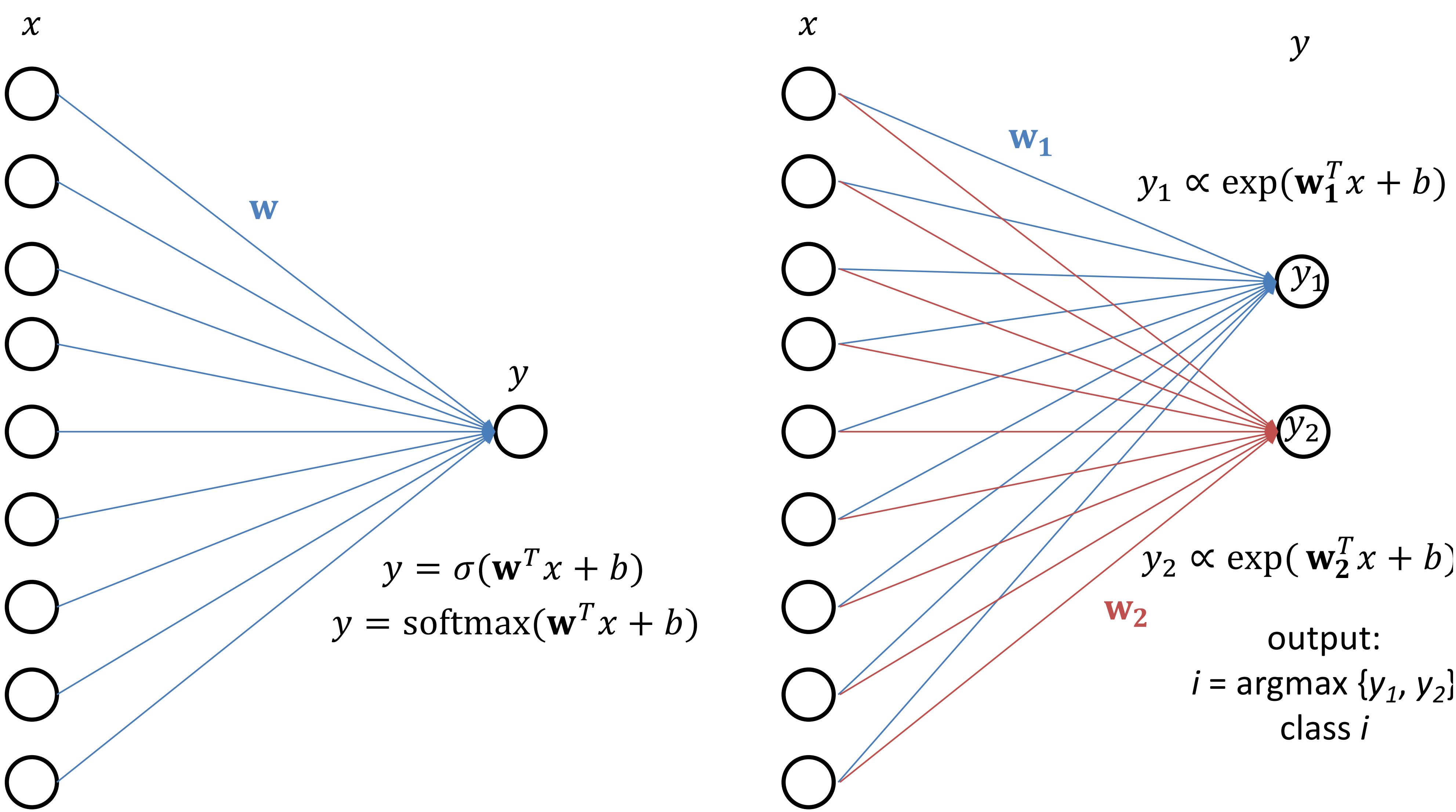
$$y = \mathbf{w}^T x + b$$

output:
if $y > 0$: class 1
else: class 2

Linear Regression/Perceptron: A Per-Class View

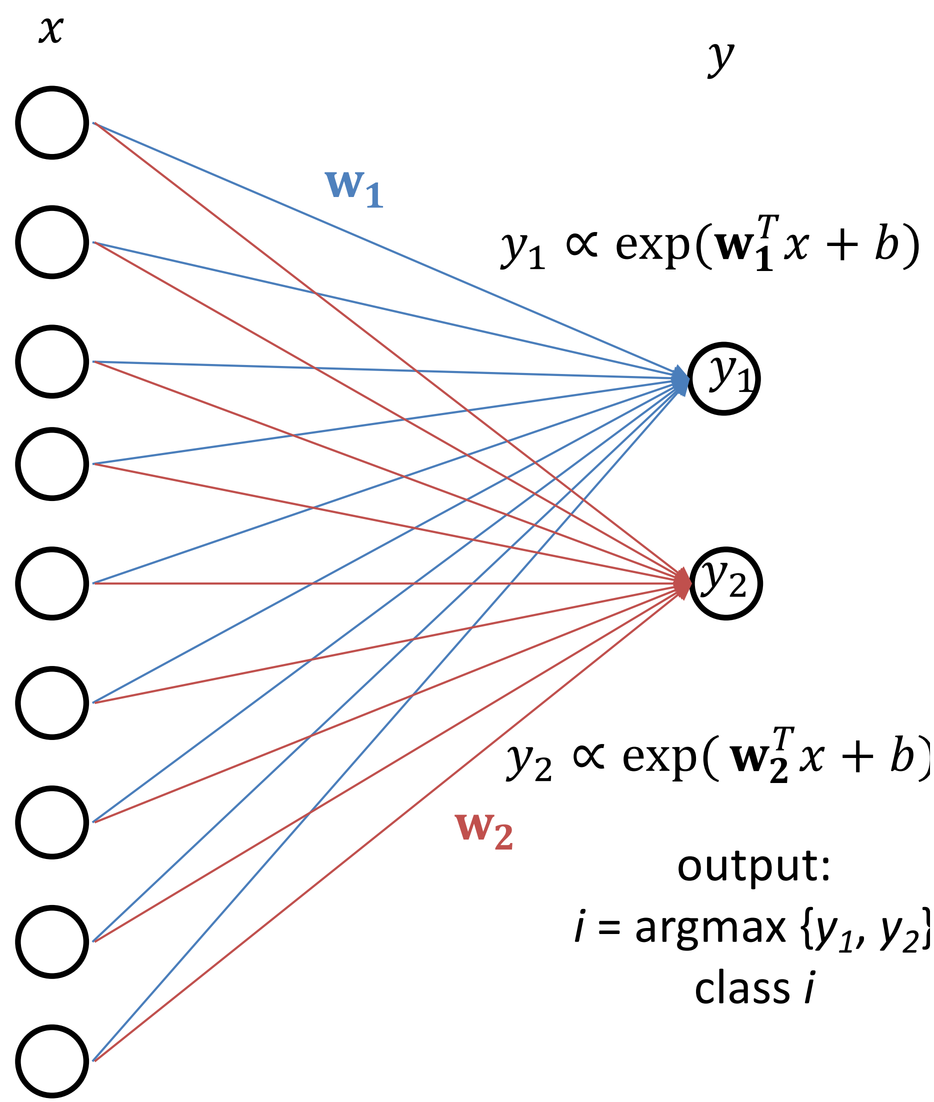


Logistic Regression/Classification



Logistic Regression/Classification

Q: Why didn't our maxent formulation from last class have multiple weight vectors?

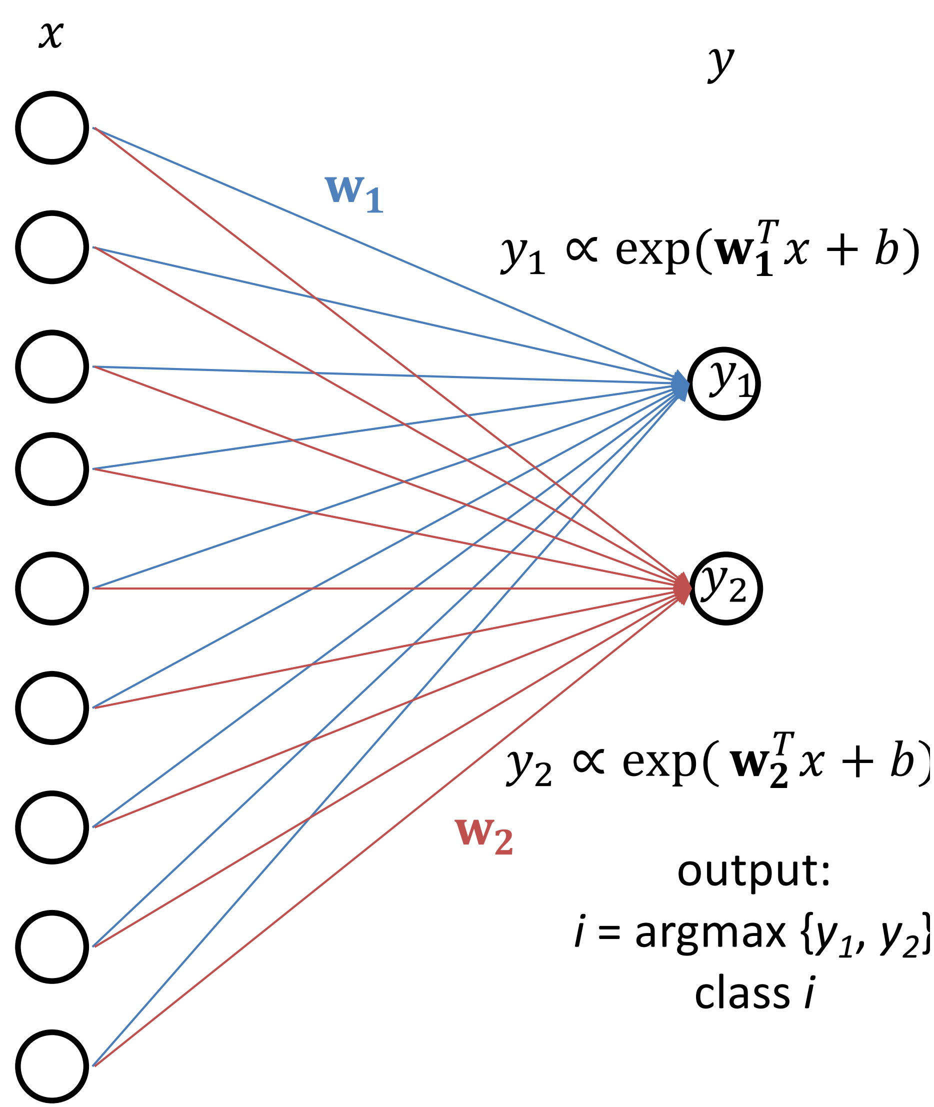


output:
i = argmax $\{y_1, y_2\}$
class i

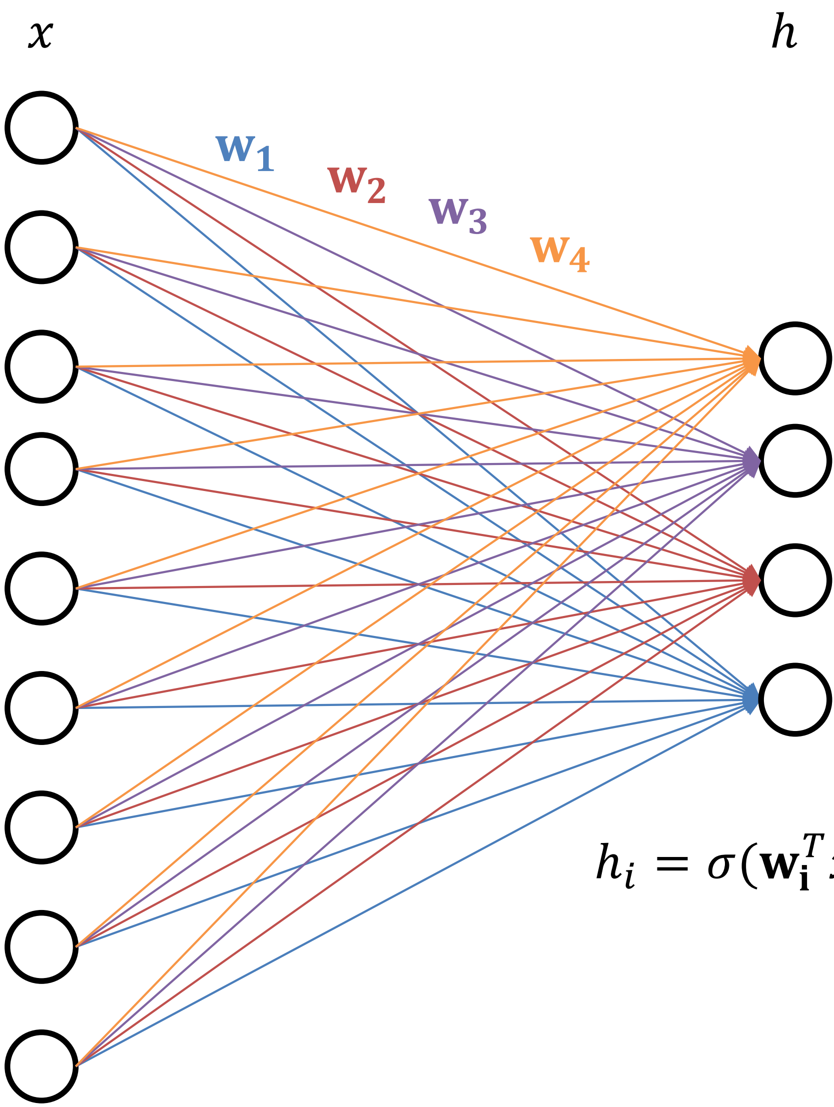
Logistic Regression/Classification

Q: Why didn't our maxent formulation from last class have multiple weight vectors?

A: Implicitly it did. Our formulation was
$y \propto \exp(w^T f(x, y))$



Stacking Logistic Regression

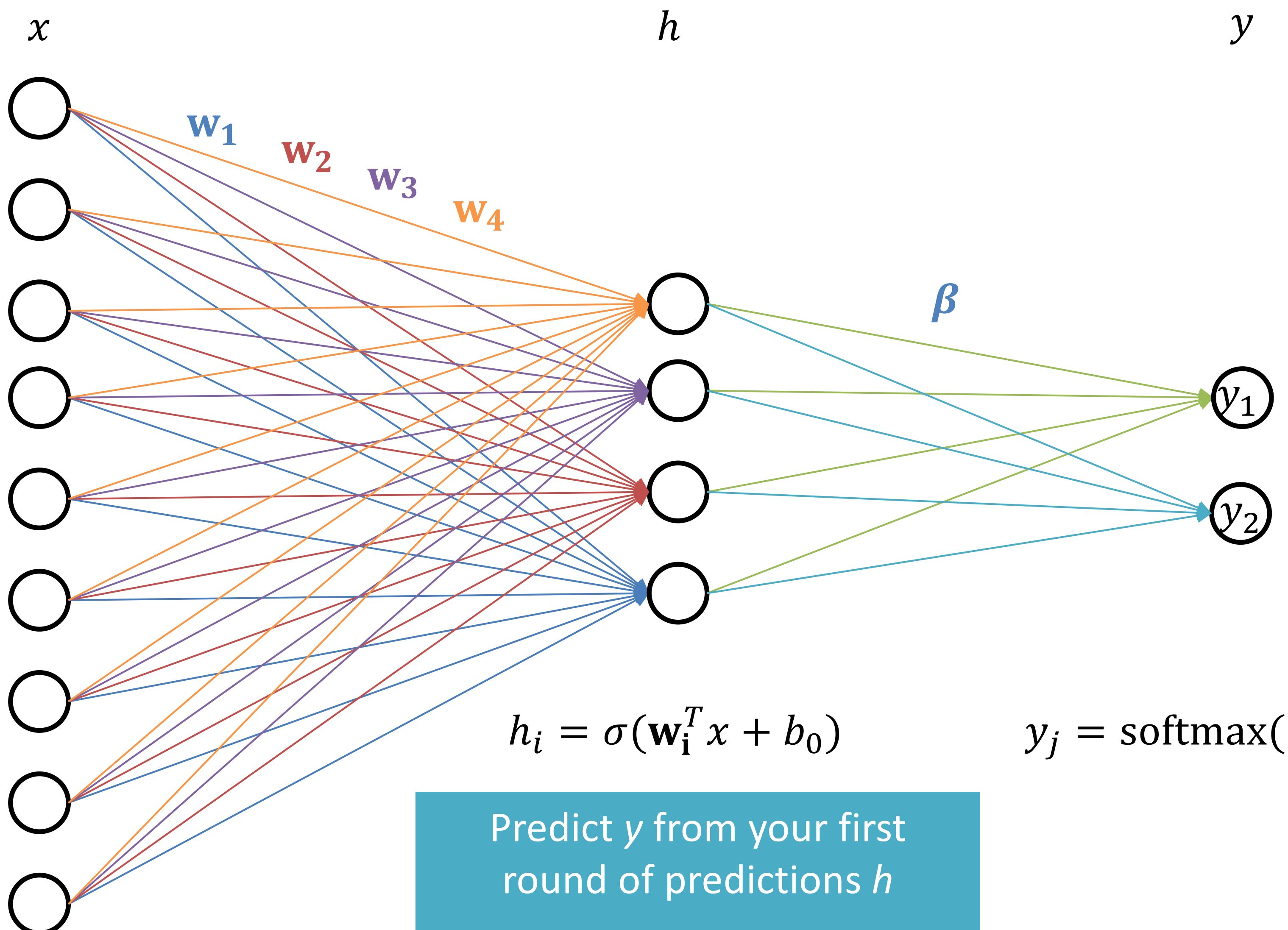


$$h_i = \sigma(\mathbf{w}_\mathbf{i}^T x + b_0)$$

Goal: you still want to predict y

Idea: Can making an initial round of separate (independent) *binary* predictions *h* help?

Stacking Logistic Regression



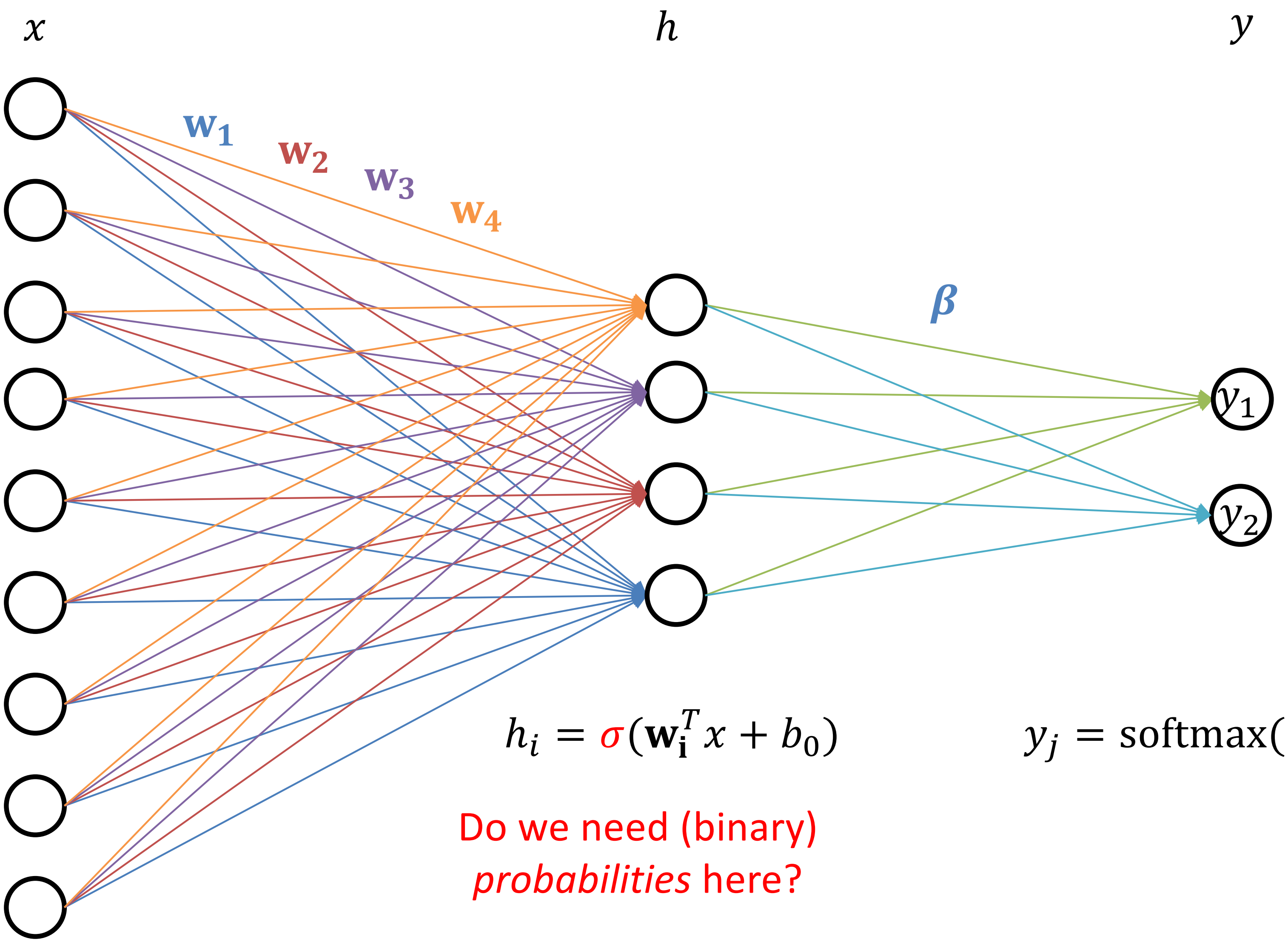
$$h_i = \sigma(\mathbf{w_i}^T x + b_0)$$

$$y_j = \text{softmax}(\boldsymbol{\beta}_\mathbf{j}^T h + b_1)$$

Predict *y* from your first round of predictions *h*

Idea: data/signal compression

Stacking Logistic Regression

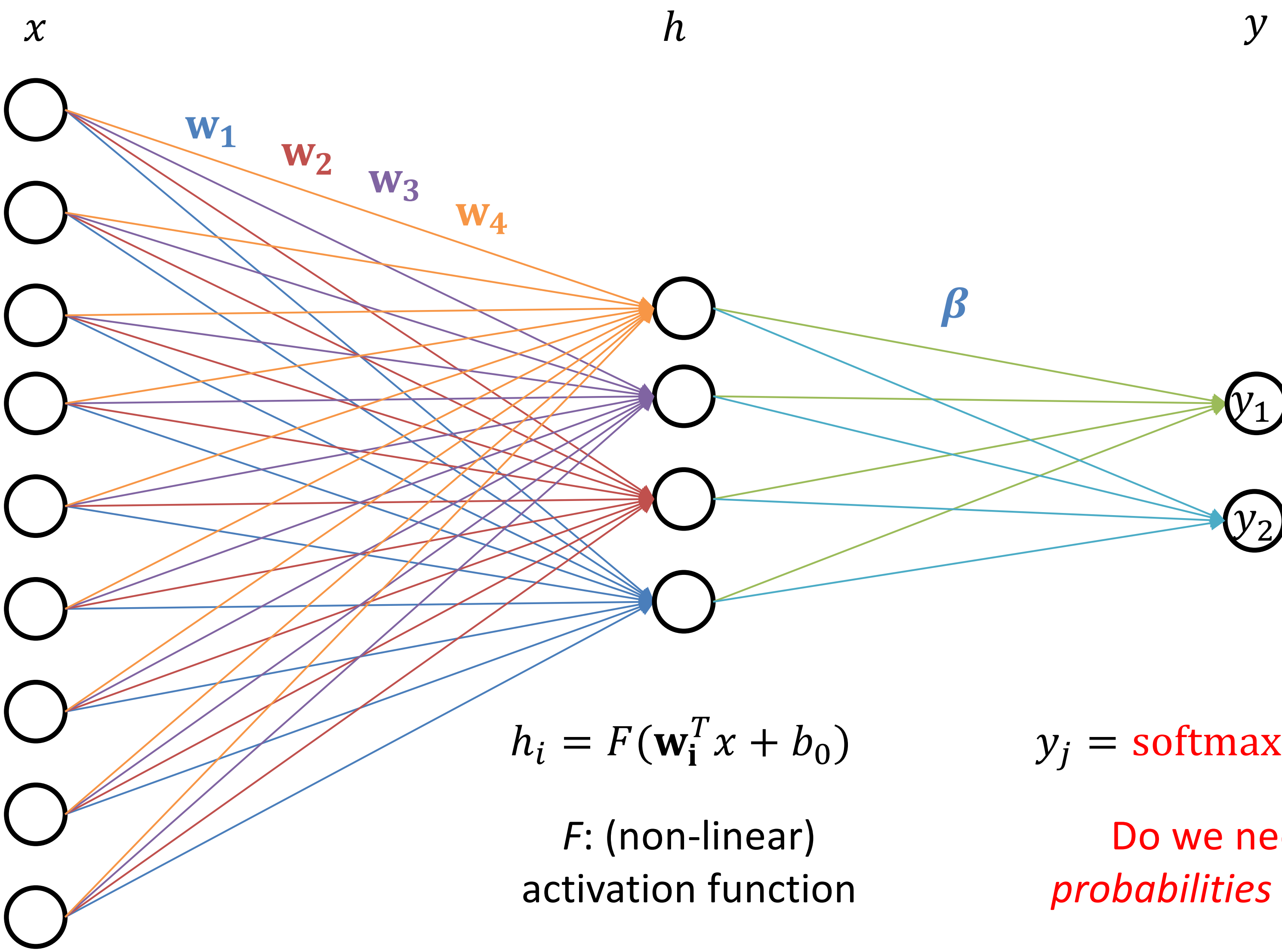


$h_i = \sigma(\mathbf{w_i}^T x + b_0)$

$y_j = \text{softmax}(\boldsymbol{\beta}_\mathbf{j}^T h + b_1)$

Do we need (binary) *probabilities* here?

Stacking Logistic Regression



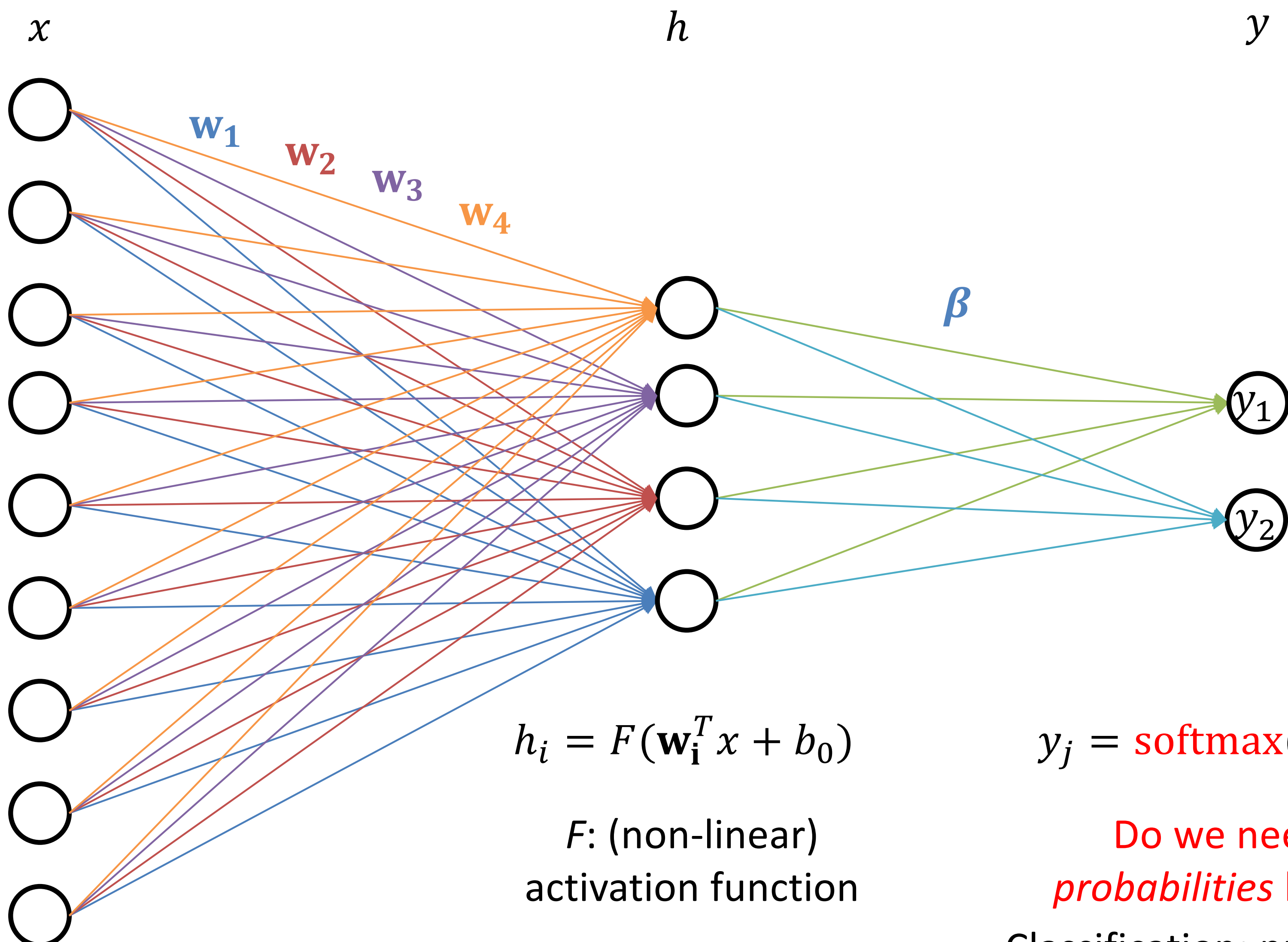
$h_i = F(\mathbf{w}_\mathbf{i}^T x + b_0)$

F: (non-linear) activation function

$y_j = \text{softmax}(\boldsymbol{\beta}_\mathbf{j}^T h + b_1)$

Do we need *probabilities* here?

Stacking Logistic Regression



$$h_i = F(\mathbf{w_i}^T x + b_0)$$

F: (non-linear) activation function

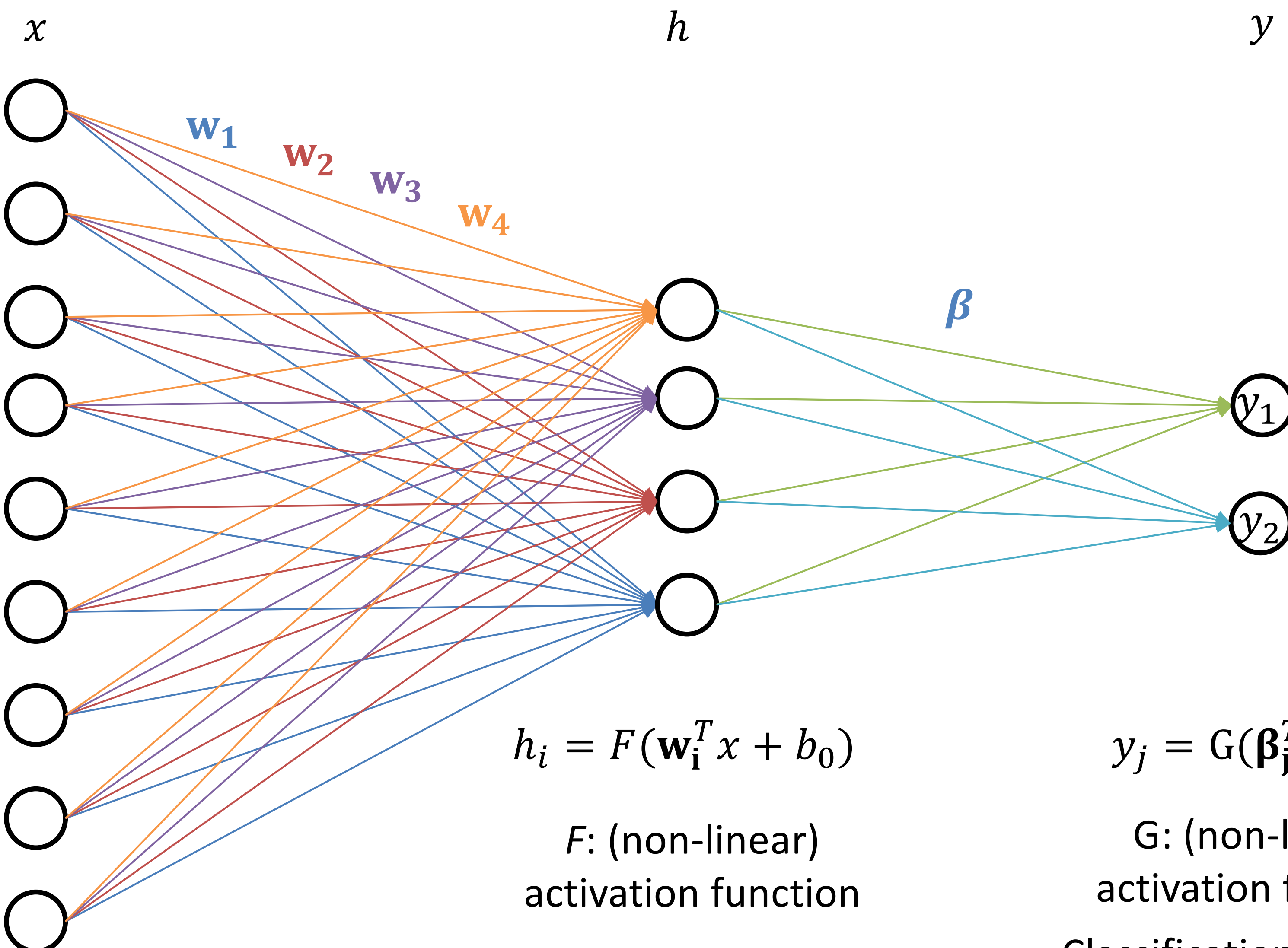
$$y_j = \text{softmax}(\boldsymbol{\beta}_\mathbf{j}^T h + b_1)$$

Do we need *probabilities* here?

Classification: probably

Regression: not really

Stacking Logistic Regression



$$h_i = F(\mathbf{w}_\mathbf{i}^T x + b_0)$$

F: (non-linear) activation function

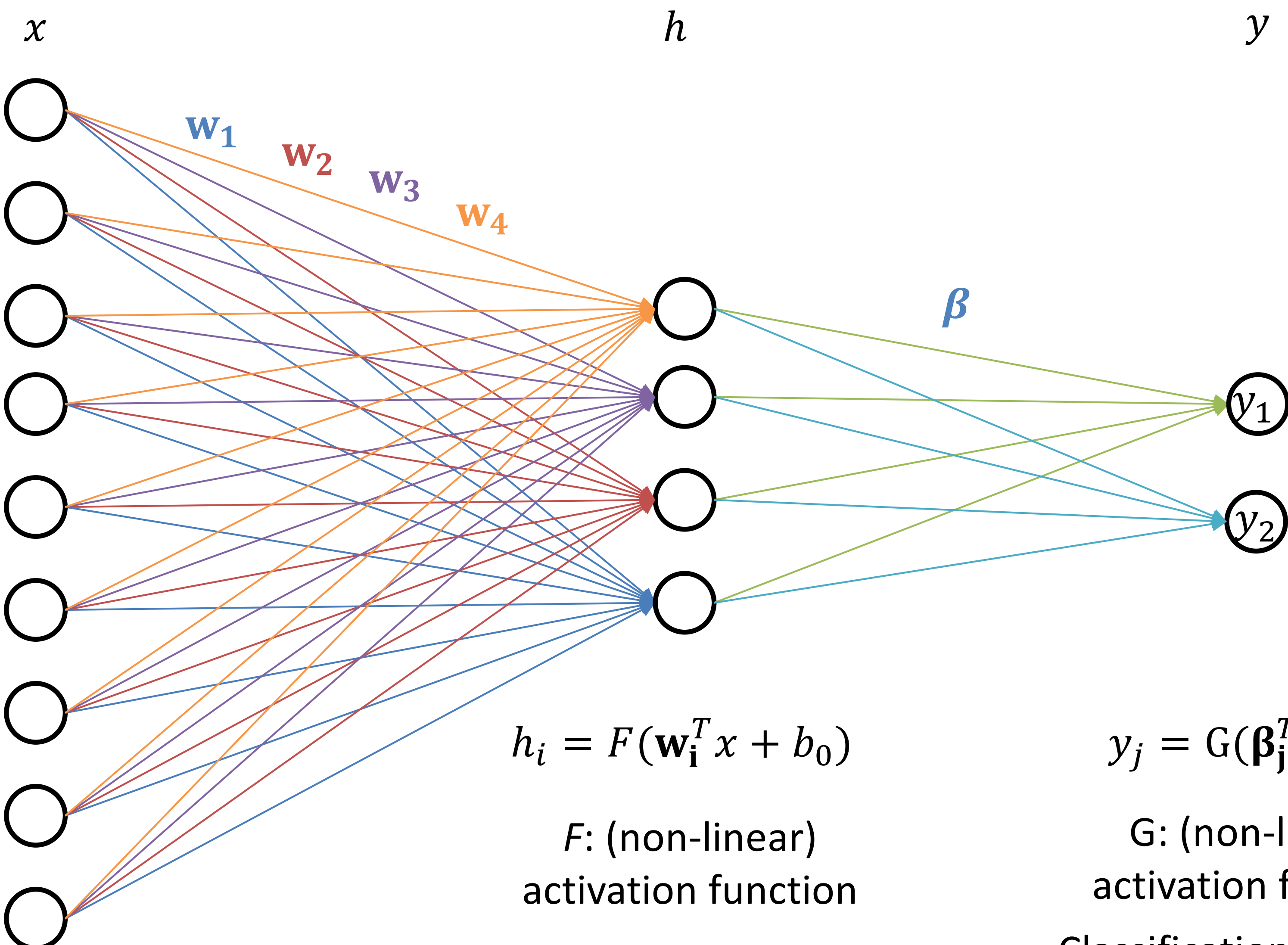
$$y_j = \mathrm{G}(\boldsymbol{\beta}_\mathbf{j}^T h + b_1)$$

G: (non-linear) activation function

Classification: softmax

Regression: identity

Multilayer Perceptron, a.k.a. Feed-Forward Neural Network



$$h_i = F(\mathbf{w_i}^T x + b_0)$$

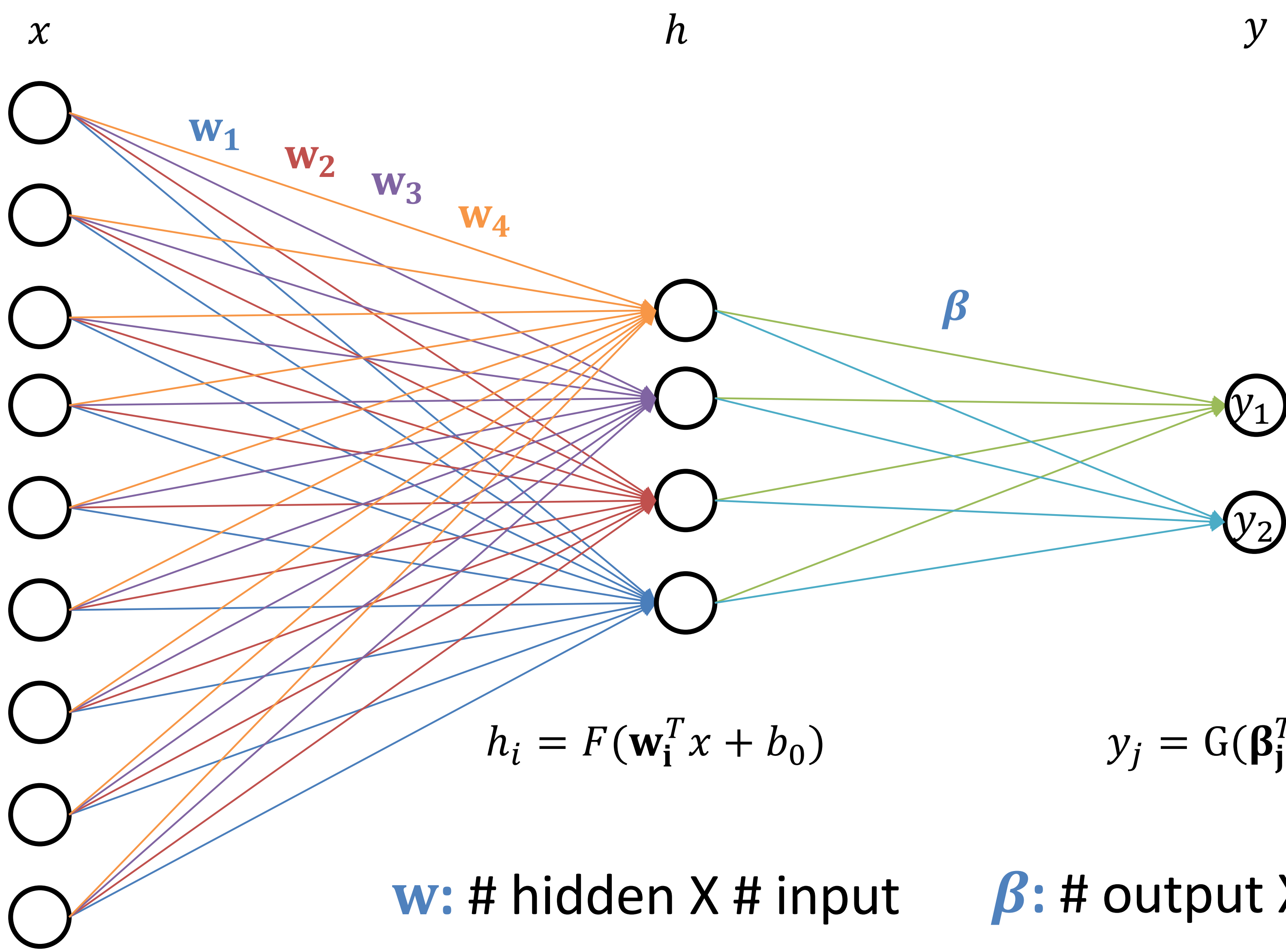
F: (non-linear) activation function

$$y_j = \mathrm{G}(\boldsymbol{\beta}_\mathbf{j}^T h + b_1)$$

G: (non-linear) activation function

Classification: softmax
Regression: identity

Feed-Forward Neural Network



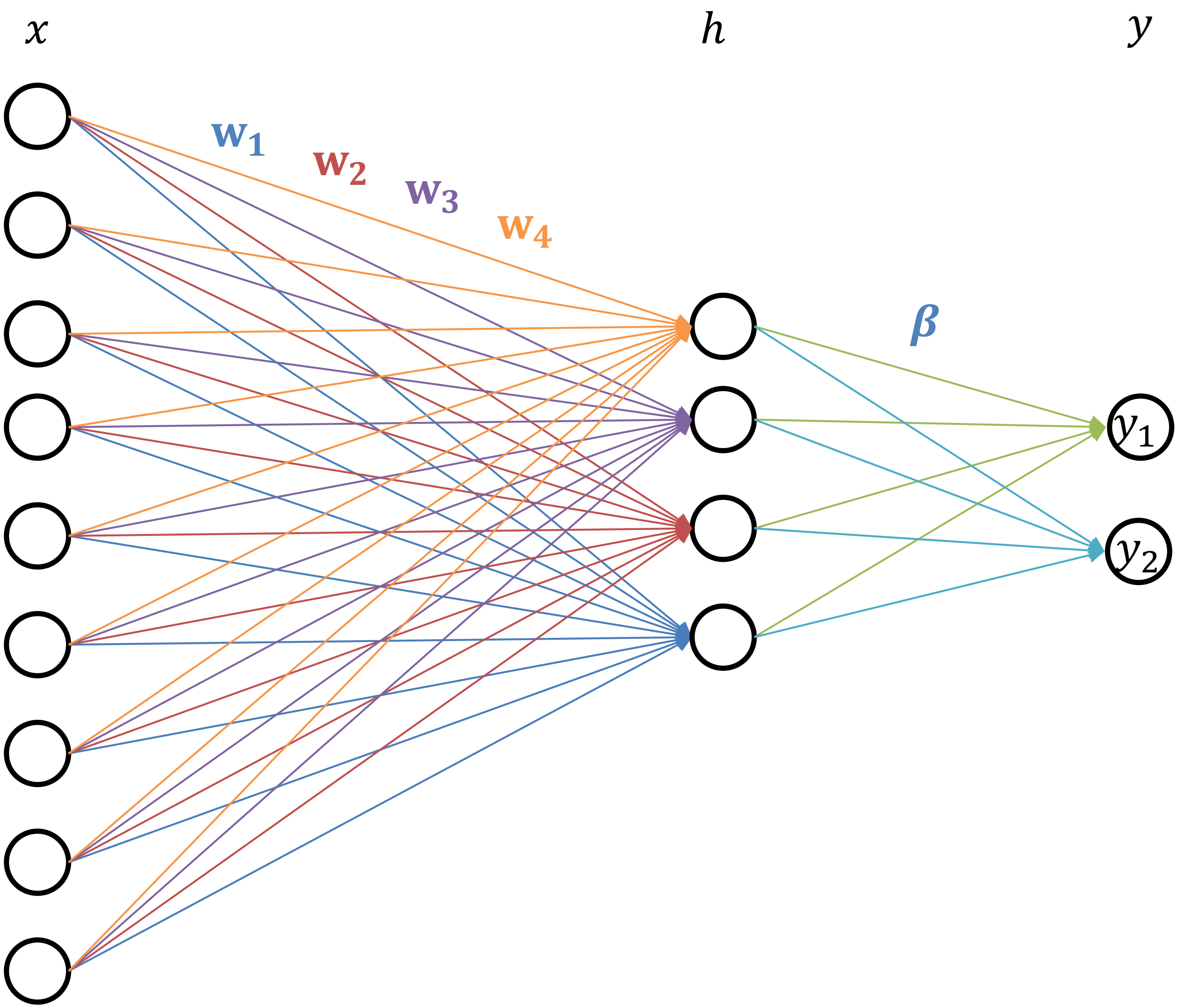
$h_i = F(\mathbf{w}_\mathbf{i}^T x + b_0)$

$y_j = \mathrm{G}(\boldsymbol{\beta}_\mathbf{j}^T h + b_1)$

$\mathbf{w}$: # hidden X # input

$\boldsymbol{\beta}$: # output X # hidden

Why Non-Linear?

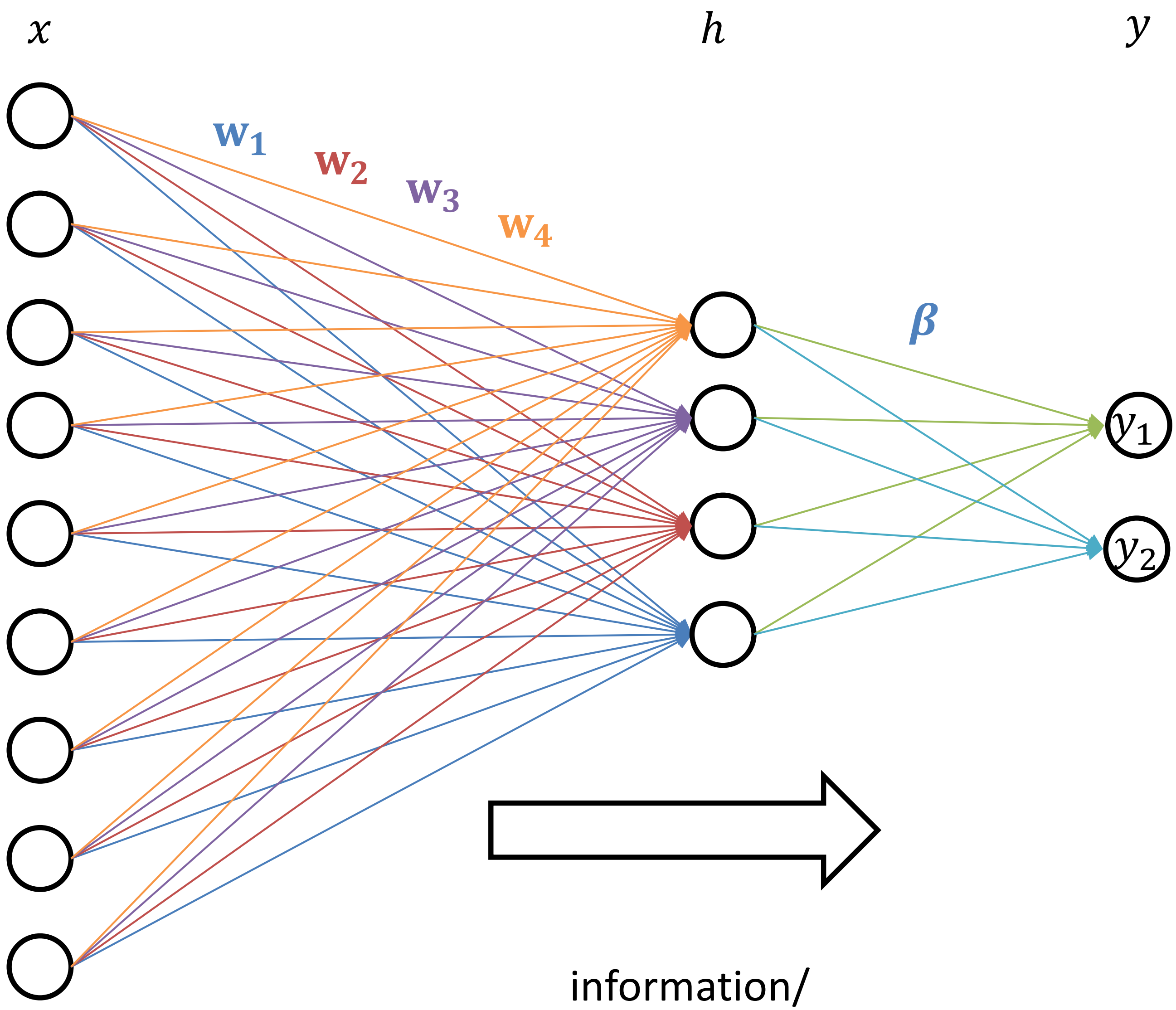


$$y_j = \mathrm{G}\left(\boldsymbol{\beta}_j^T h + b_1\right)$$

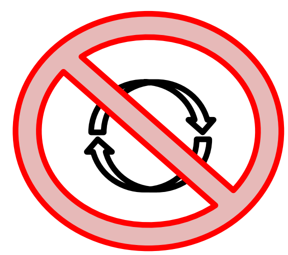
⇩

$$y_j = G\left(\beta_j^T \left(F\left(w_i^T x + b_0\right)\right)_i\right)$$

Feed-Forward

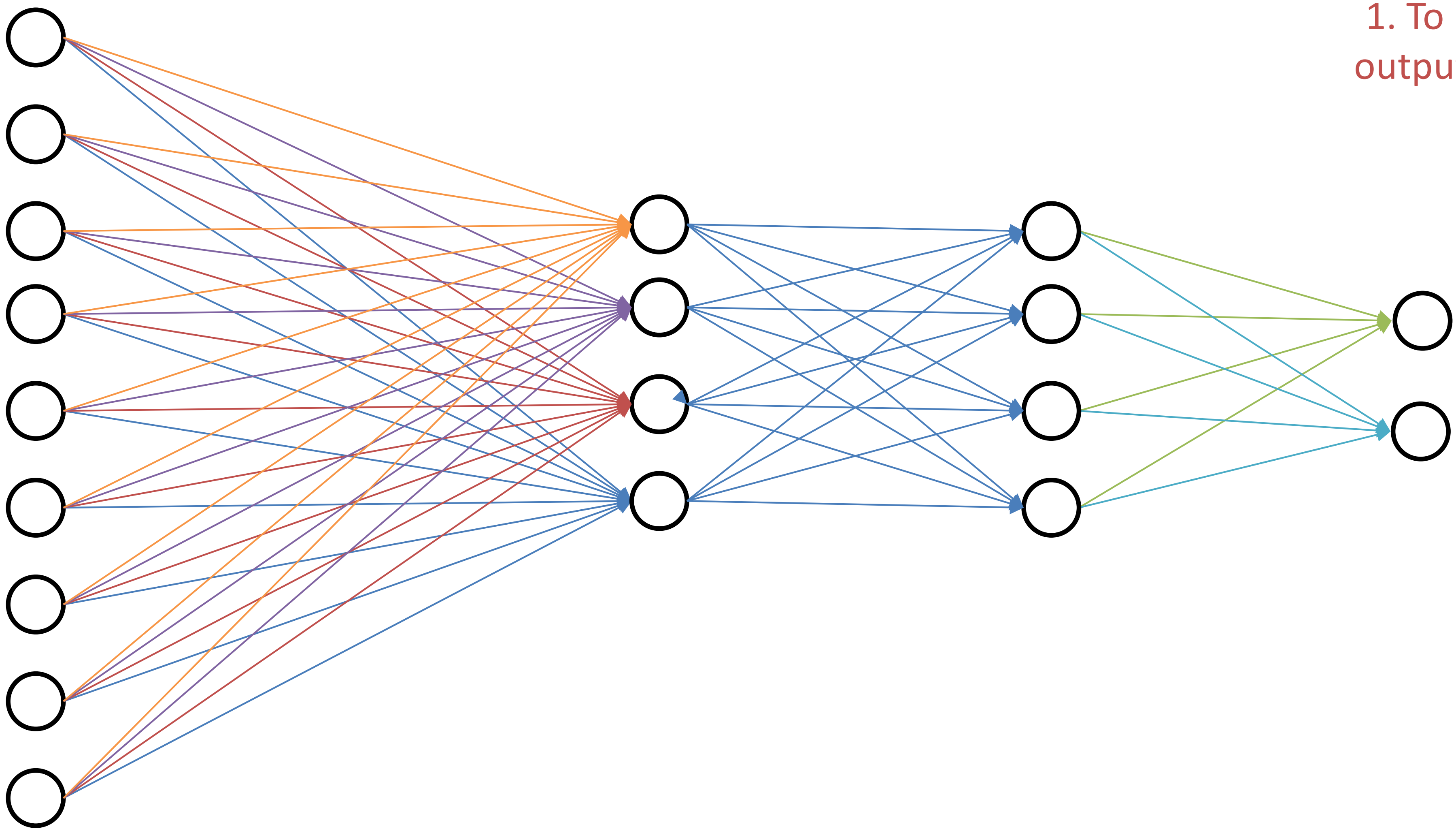


information/
computation flow



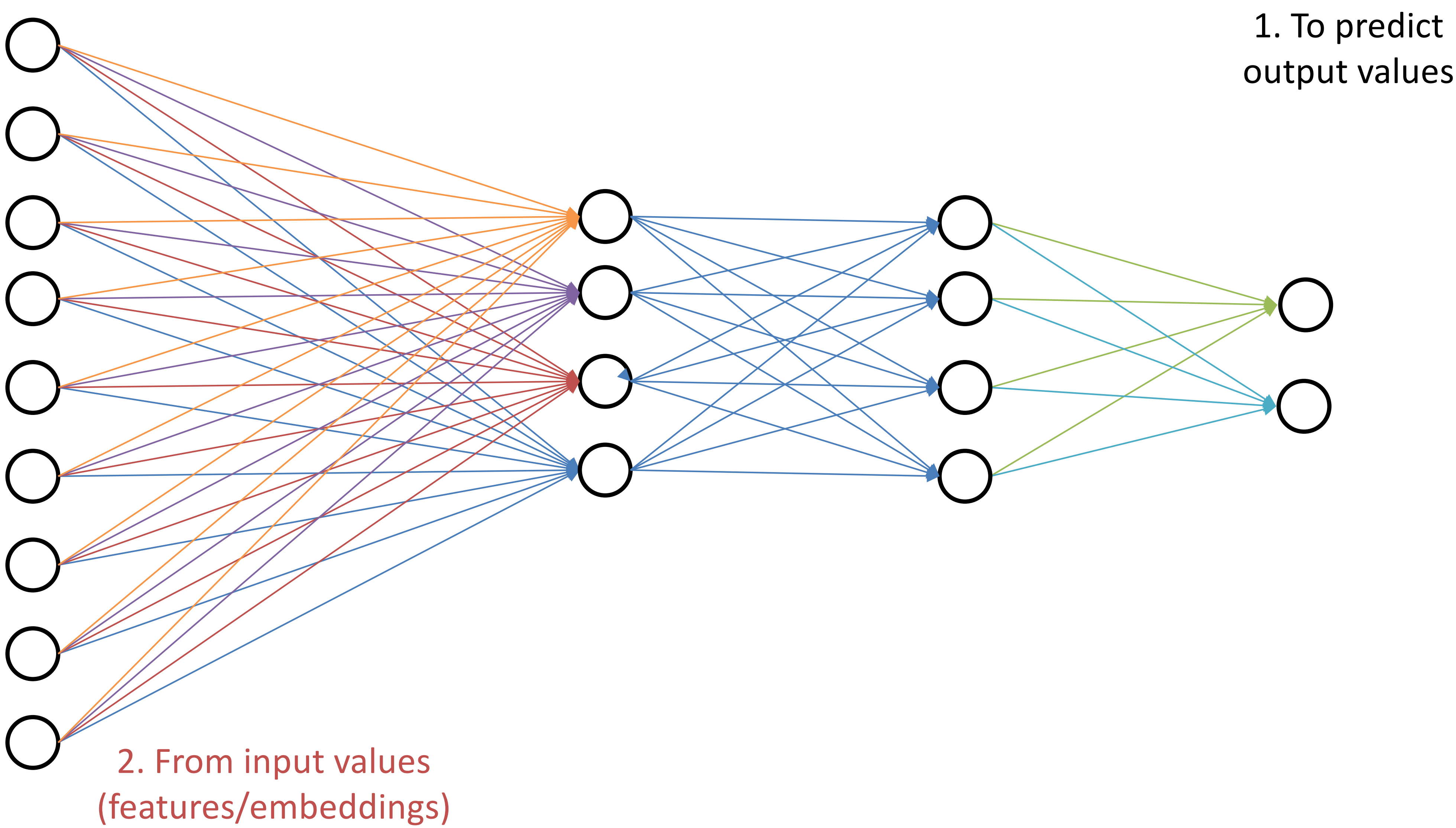
no self-loops
(recurrence/reuse of weights)

A Neural Network is a Machine Learning System…

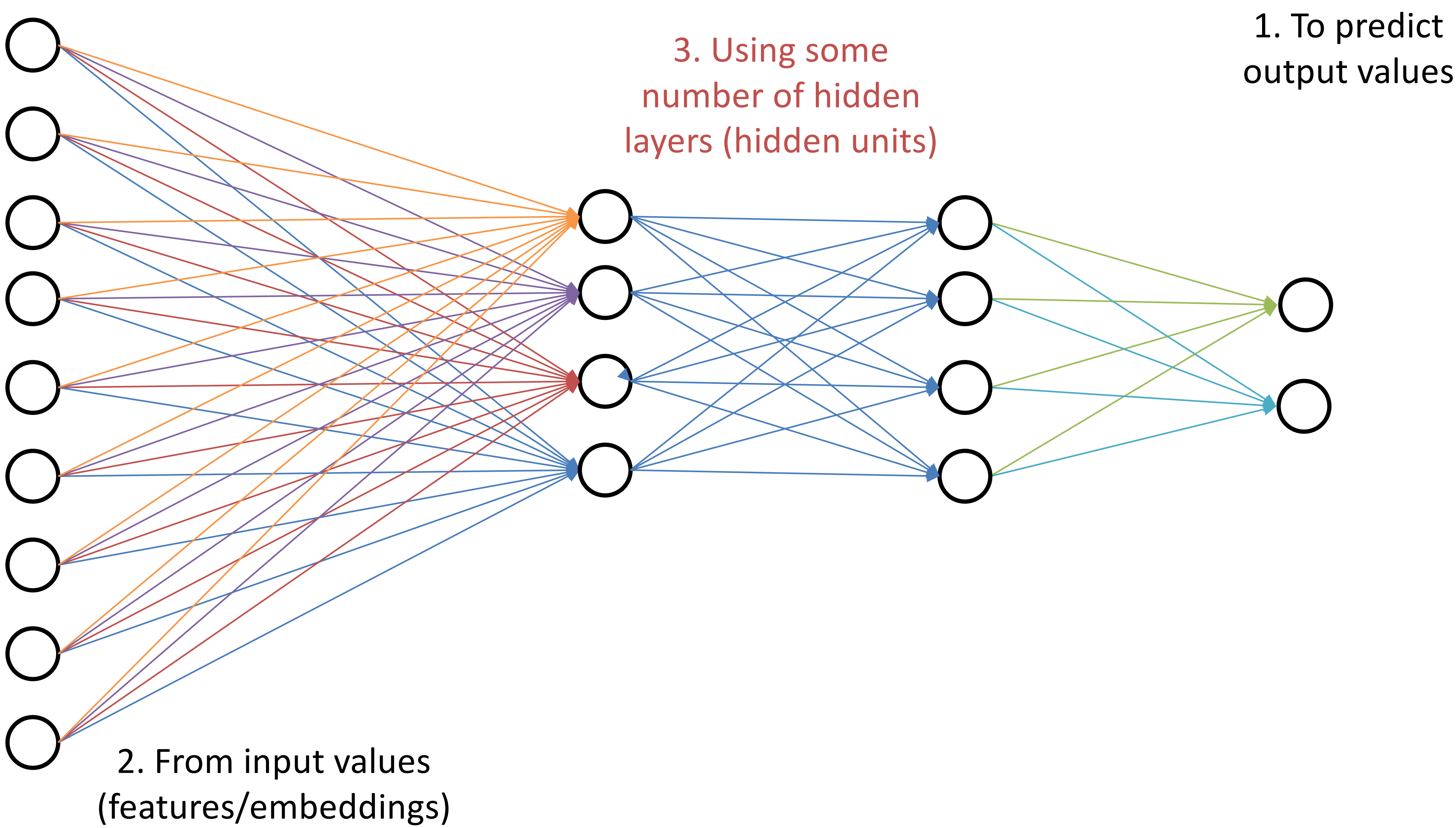


1. To predict output values

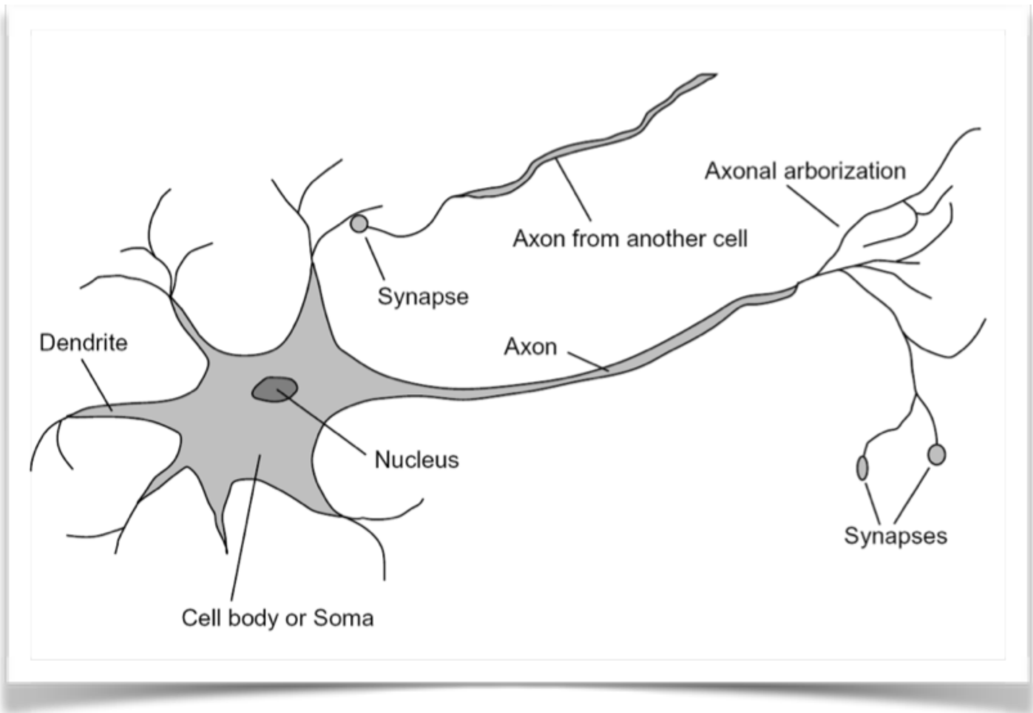
A Neural Network is a Machine Learning System…



A Neural Network is a Machine Learning System…



Why “Neural?”



argue from neuroscience perspective

neurons (in the brain) receive input and “fire”
when sufficiently excited/activated

Image courtesy Hamed Pirsiavash

Universal Function Approximator

Theorem [Kurt Hornik et al., 1989]: Let F be a continuous function on a bounded subset of D-dimensional space. Then there exists a two-layer network G with finite number of hidden units that approximates F arbitrarily well. For all x in the domain of F, $|F(x) – G(x) |< \varepsilon$

“a two-layer network can approximate any function”

Going from one to two layers dramatically improves the representation power of the network

Slide courtesy Hamed Pirsiavash

How Deep Can They Be?

So many choices:

Architecture

# of hidden layers

# of units per hidden layer

Computational Issues:

Vanishing gradients

Gradients shrink as one moves away from the output layer

Convergence is slow

Opportunities:

Training deep networks is an active area of research

Layer-wise initialization (perhaps using unsupervised data)

Engineering: GPUs to train on massive labelled datasets

Slide courtesy Hamed Pirsiavash

Some Results: Digit Classification

logistic regression

simple feed forward

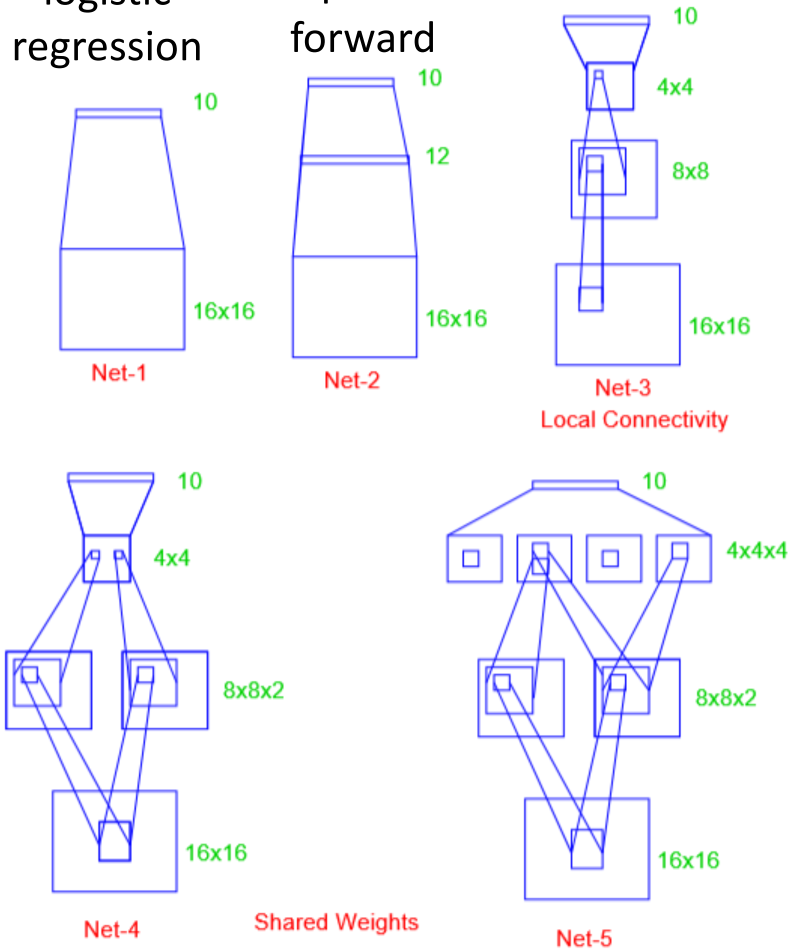


FIGURE 11.10. *Architecture of the five networks used in the ZIP code example.*

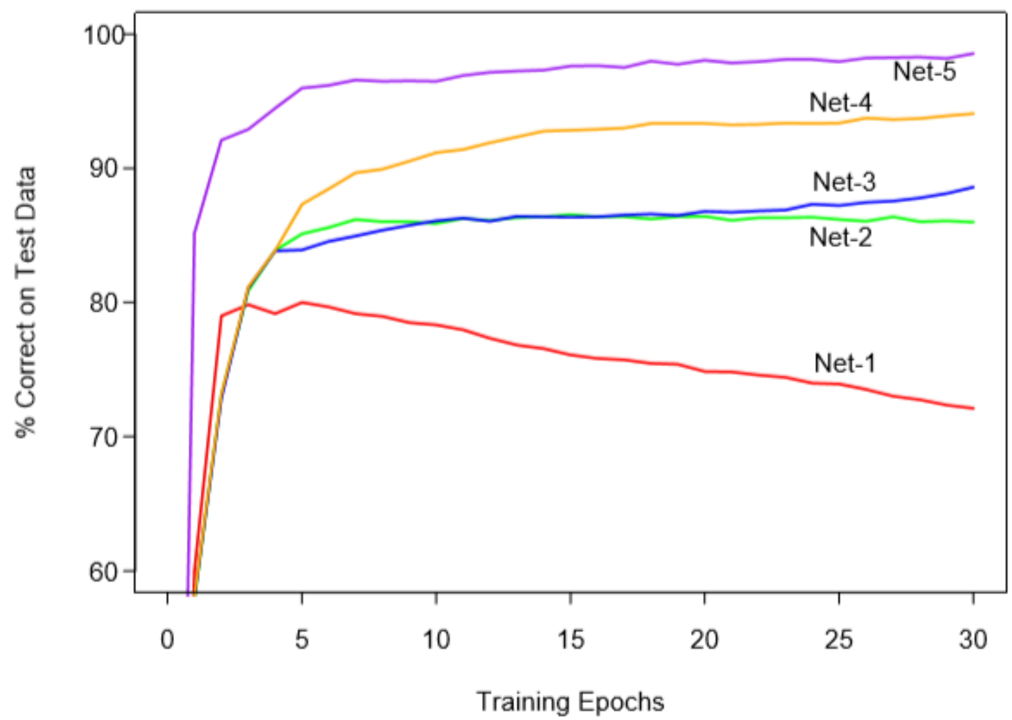


FIGURE 11.11. *Test performance curves, as a function of the number of training epochs, for the five networks of Table 11.1 applied to the ZIP code data.*

(similar to MNIST in A2, but not exactly the same)

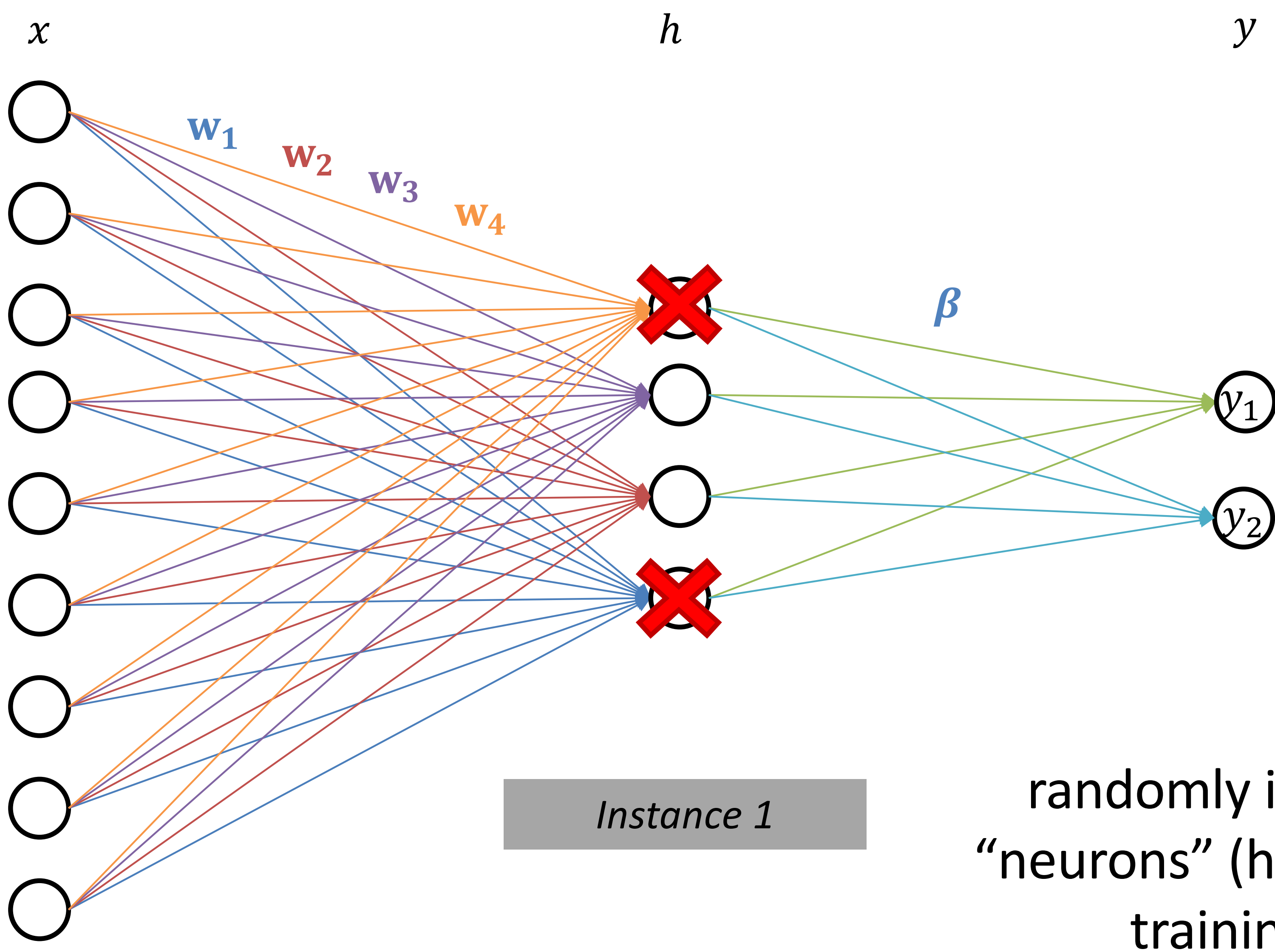
Tensorflow Playground

http://playground.tensorflow.org

Experiment with small (toy) data neural networks in your browser

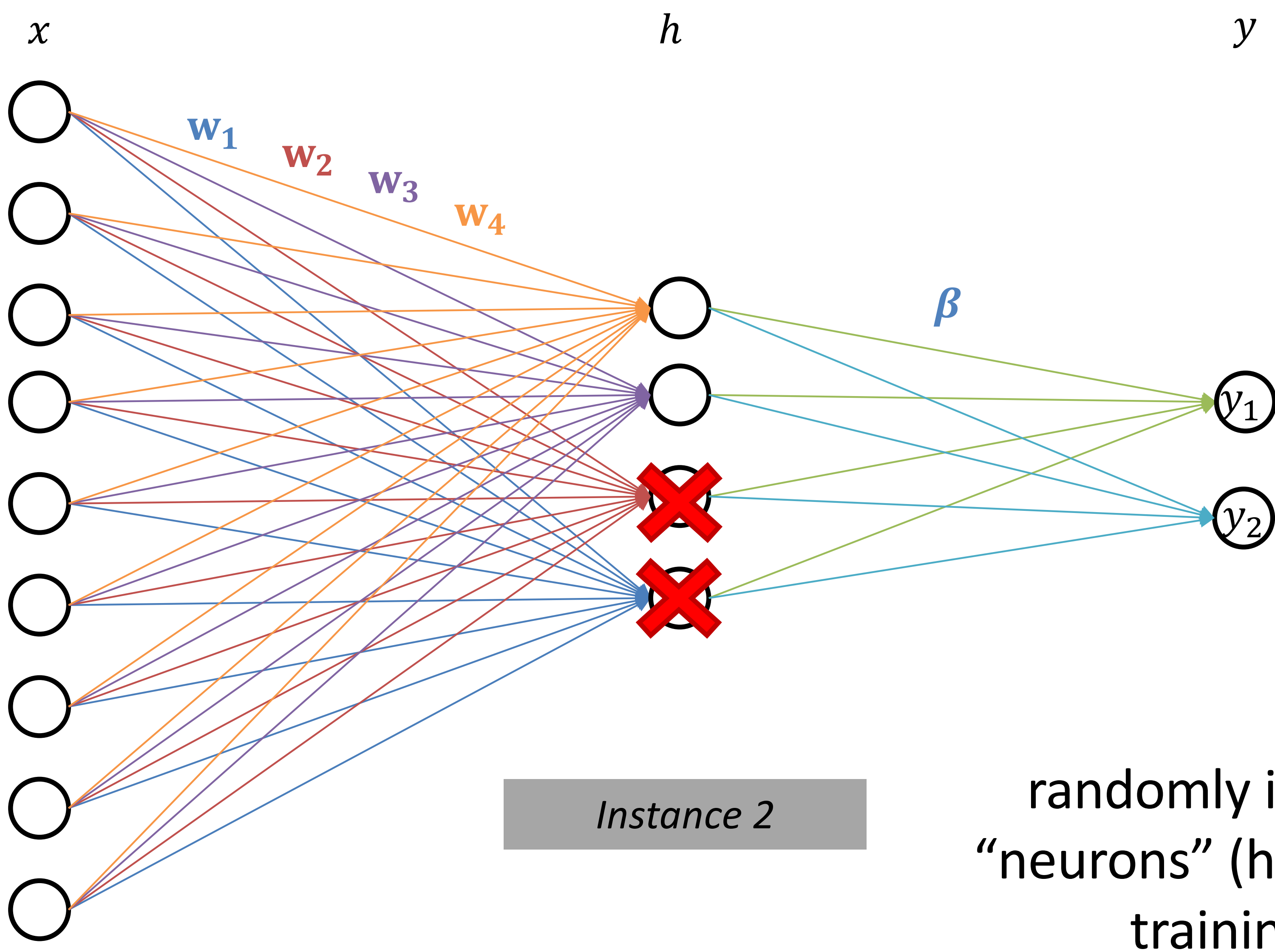
Feel free to use this to gain an intuition

Dropout: Regularization in Neural Networks



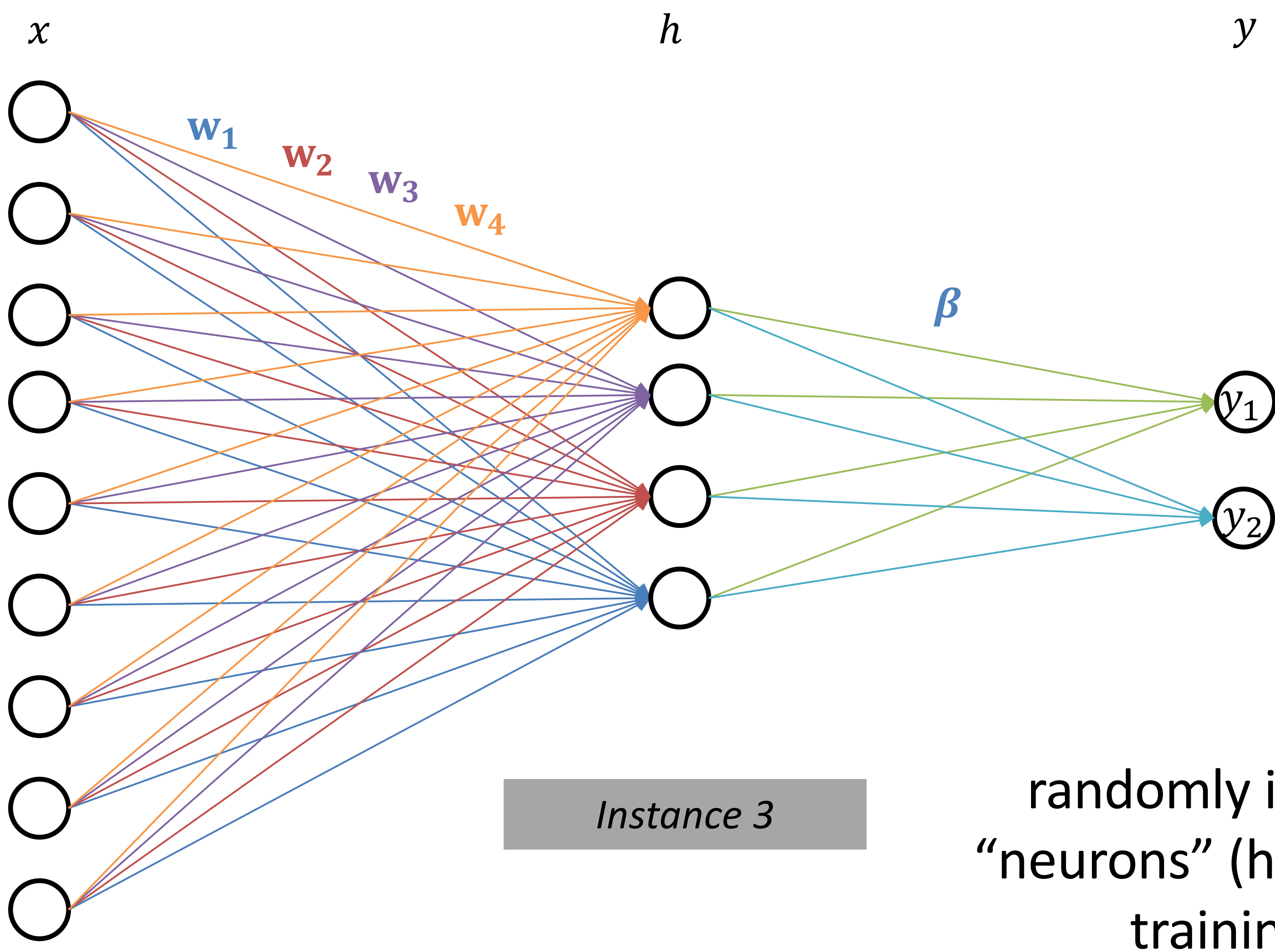
randomly ignore "neurons" (h_i) during training

Dropout: Regularization in Neural Networks



randomly ignore "neurons" (h_i) during training

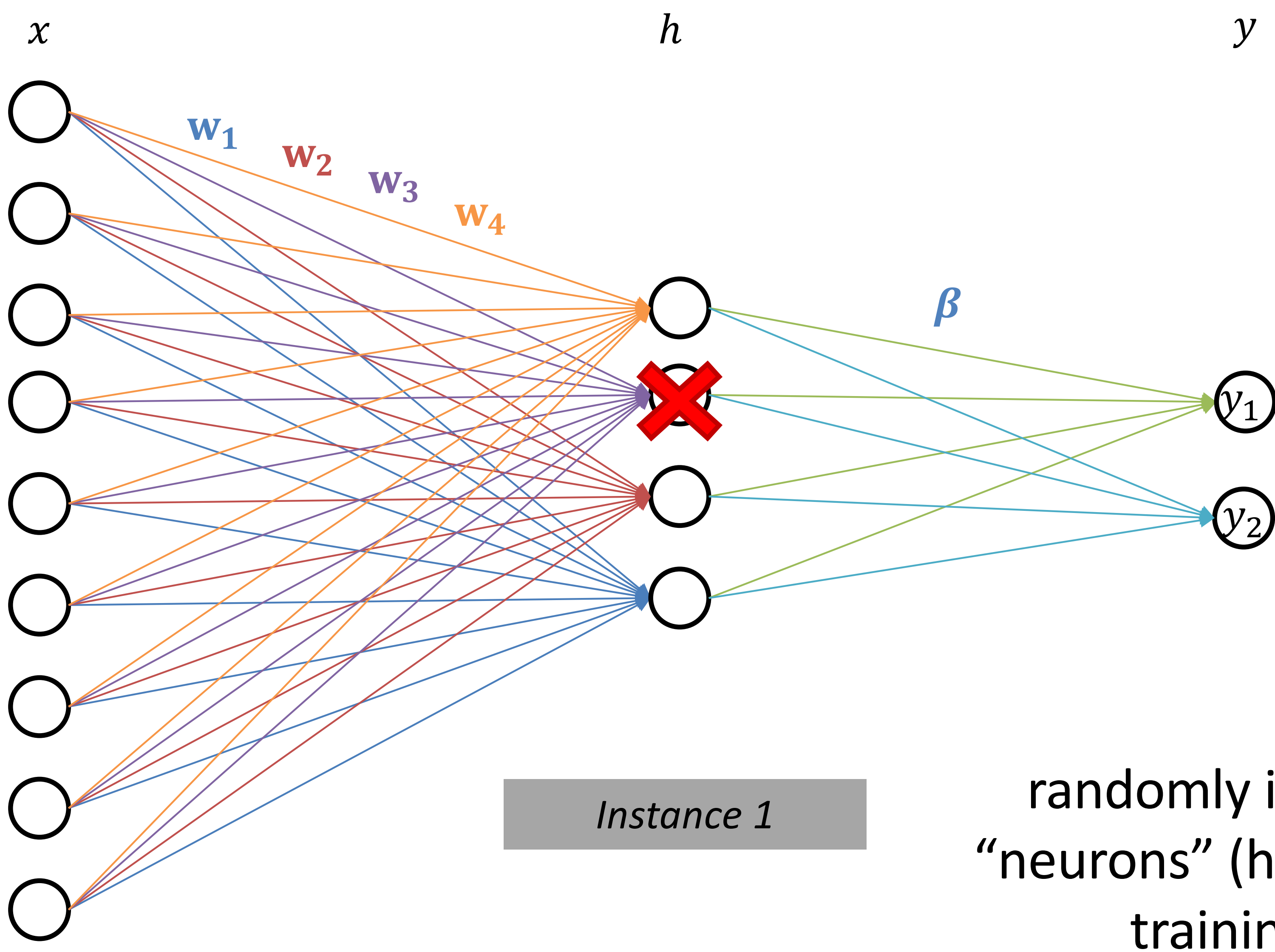
Dropout: Regularization in Neural Networks



Instance 3

randomly ignore "neurons" (h_i) during training

Dropout: Regularization in Neural Networks

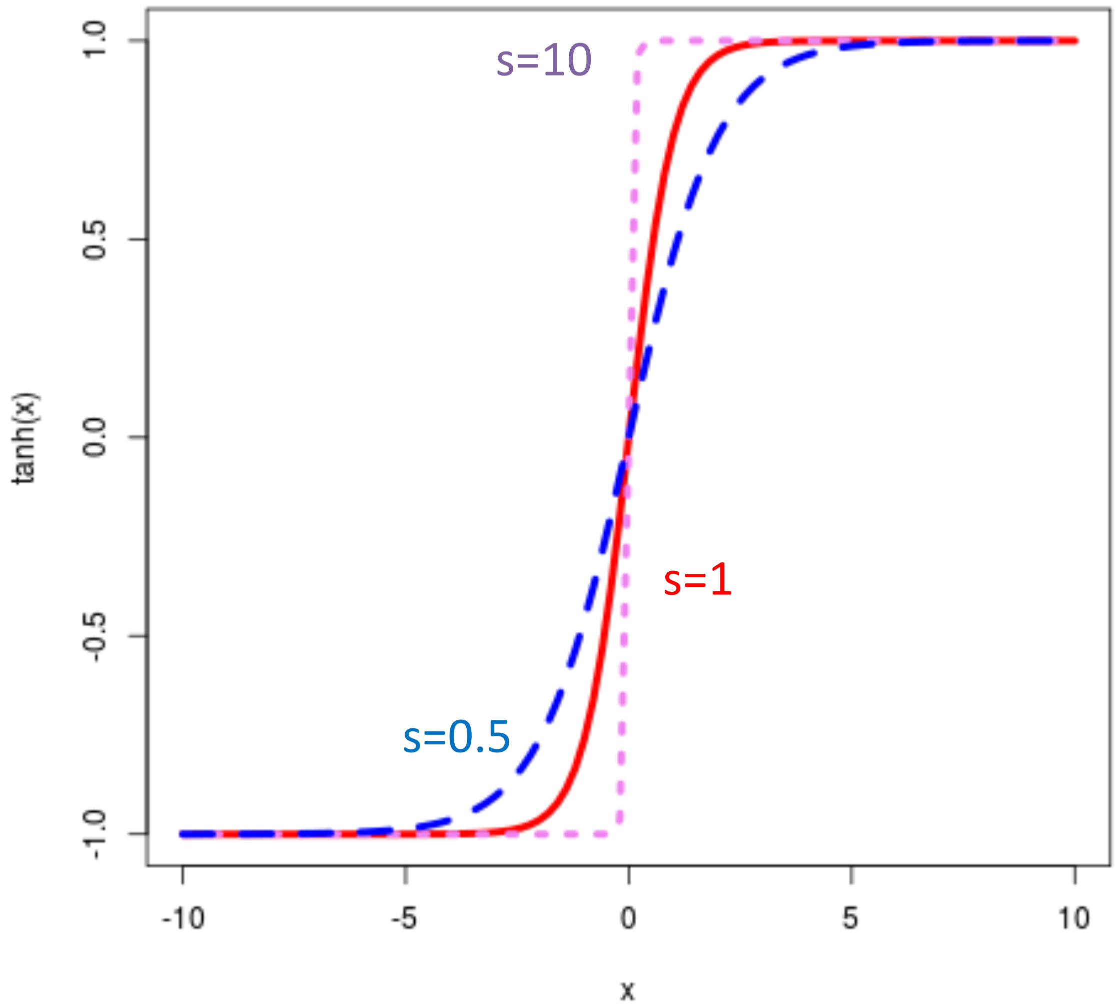


randomly ignore "neurons" (h_i) during training

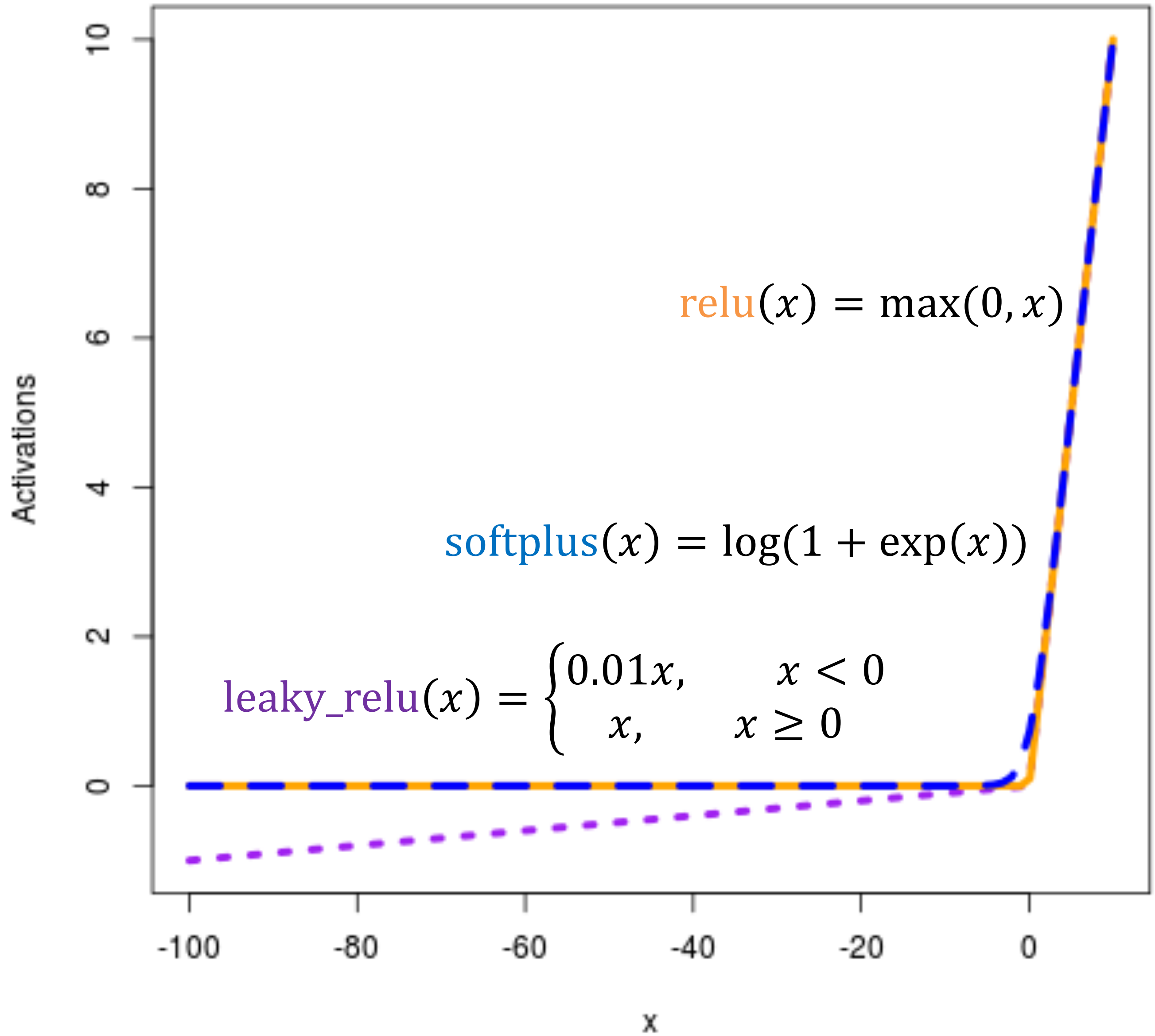
tanh Activation

$$\tanh_s(x) = \frac{2}{1 + \exp(-2 * s * x)} - 1$$

$$= 2\sigma_s(x) - 1$$



Rectifiers Activations



Outline

Neural networks: non-linear classifiers

Learning weights: backpropagation of error

Autodifferentiation (in reverse mode)

Empirical Risk Minimization

Cross entropy loss

$$\ell^{\text{xent}}\left(\overrightarrow{y^*}, y\right) = -\sum_k \overrightarrow{y^*}[k] \log p(y = k)$$

mean squared error/L2 loss

$$\ell^{\text{L2}}(y^*, y) = (y^* - y)\text{^}2$$

squared expectation loss

$$\ell^{\text{sq-expt}}\left(\overrightarrow{y^*}, y\right) = \left|\overrightarrow{y^*} - p(y)\right|_2^2$$

hinge loss

$$\ell^{\text{hinge}}\left(\overrightarrow{y^*}, y\right) = \max\left\{0, 1 + \max_{j \neq y^*}\left(y[j] - \overrightarrow{y^*}[j]\right)\right\}$$

Gradient Descent: Backpropagate the Error

Set $t = 0$

Pick a starting value θ_t

Until converged:

  for example(s) i:

  1. Compute loss l on x_i
  2. Get gradient $g_t = l'(x_i)$
  3. Get scaling factor ρ_t
  4. Set $\theta_{t+1} = \theta_t - \rho_t * g_t$
  5. Set $t \mathrel{+}= 1$

(mini)batch

epoch

epoch: a single run over all training data

(mini-)batch: a run over a subset of the data

Flavors of Gradient Descent

“Online”

Set t = 0
Pick a starting value θ_t
Until converged:

for example i in full data:

1. Compute loss l on x_i
2. Get gradient
   $g_t = l'(x_i)$
3. Get scaling factor ρ_t
4. Set $\theta_{t+1} = \theta_t - \rho_t * g_t$
5. Set t += 1

done

“Minibatch”

Set t = 0
Pick a starting value θ_t
Until converged:

get batch $B \subset$ full data
set $g_t = 0$
for example(s) i in B:

1. Compute loss l on x_i
2. Accumulate gradient
   $g_t \mathrel{+}= l'(x_i)$

done
Get scaling factor ρ_t
Set $\theta_{t+1} = \theta_t - \rho_t * g_t$
Set t += 1

“Batch”

Set t = 0
Pick a starting value θ_t
Until converged:

set $g_t = 0$
for example(s) i in full data:

1. Compute loss l on x_i
2. Accumulate gradient
   $g_t \mathrel{+}= l'(x_i)$

done
Get scaling factor ρ_t
Set $\theta_{t+1} = \theta_t - \rho_t * g_t$
Set t += 1

Gradients for Feed Forward Neural Network

$$y_k = \sigma\left(\beta_k^T \underbrace{\left(\sigma\left(w_j^T x + b_0\right)\right)_j}_{h\text{: a vector}}\right)$$

$$\mathcal{L} = -\sum_k \overrightarrow{y^*}[k] \log y_k$$

$$\frac{\partial \mathcal{L}}{\partial \beta_{kj}} = \frac{-1}{y_{y^*}} \frac{\partial y_{y^*}}{\partial \beta_{kj}}$$

$$\frac{\partial \mathcal{L}}{\partial w_{jl}}$$

Gradients for Feed Forward Neural Network

$$y_k = \sigma\left(\beta_k^T \underbrace{\left(\sigma\left(w_j^T x + b_0\right)\right)_j}_{h\text{: a vector}}\right)$$

$$\mathcal{L} = -\sum_k \overrightarrow{y^*}[k] \log y_k$$

$$\frac{\partial \mathcal{L}}{\partial \beta_{kj}} = \frac{-1}{y_{y^*}} \frac{\partial y_{y^*}}{\partial \beta_{kj}} = \frac{-\sigma'\left(\beta_{y^*}^T h\right)}{\sigma\left(\beta_{y^*}^T h\right)} \frac{\partial \beta_k^T h}{\partial \beta_{kj}}$$

$$\frac{\partial \mathcal{L}}{\partial w_{jl}}$$

Gradients for Feed Forward Neural Network

$$y_k = \sigma\left(\beta_k^T \left(\sigma\left(w_j^T x + b_0\right)\right)_j\right) \qquad \mathcal{L} = -\sum_k \overrightarrow{y^*}[k] \log y_k$$

h: a vector

$$\frac{\partial \mathcal{L}}{\partial \beta_{kj}} = \frac{-1}{y_{y^*}} \frac{\partial y_{y^*}}{\partial \beta_{kj}} = \frac{-\sigma'\left(\beta_{y^*}^T h\right)}{\sigma\left(\beta_{y^*}^T h\right)} \frac{\partial \beta_k^T h}{\partial \beta_{kj}} = \frac{-\sigma'\left(\beta_{y^*}^T h\right)}{\sigma\left(\beta_{y^*}^T h\right)} \frac{\partial \sum_j \beta_{y^* j} h_j}{\partial \beta_{kj}}$$

$$\frac{\partial \mathcal{L}}{\partial w_{jl}}$$

Gradients for Feed Forward Neural Network

$$y_k = \sigma\left(\beta_k^T \underbrace{\left(\sigma\left(w_j^T x + b_0\right)\right)_j}_{h\text{: a vector}}\right) \qquad \mathcal{L} = -\sum_k \overrightarrow{y^*}[k] \log y_k$$

$$\frac{\partial \mathcal{L}}{\partial \beta_{kj}} = \frac{-1}{y_{y^*}} \frac{\partial y_{y^*}}{\partial \beta_{kj}} = \frac{-\sigma'\left(\beta_{y^*}^T h\right)}{\sigma\left(\beta_{y^*}^T h\right)} \frac{\partial \beta_k^T h}{\partial \beta_{kj}} = \frac{-\sigma'\left(\beta_{y^*}^T h\right)}{\sigma\left(\beta_{y^*}^T h\right)} \frac{\partial \sum_j \beta_{y^* j} h_j}{\partial \beta_{kj}}$$

$$= \left(1 - \sigma\left(\beta_{y^*}^T h\right)\right) h_j$$

$$\frac{\partial \mathcal{L}}{\partial w_{jl}} = \left(1 - \sigma\left(\beta_{y^*}^T h\right)\right) \left(\beta_{y^* j} \sigma'\left(w_j^T x\right) x_l\right)$$

Gradients for Feed Forward Neural Network

$$y_k = \sigma\left(\beta_k^T \underbrace{\left(\sigma\left(w_j^T x + b_0\right)\right)_j}_{h\text{: a vector}}\right)$$

$$\mathcal{L} = -\sum_k \overrightarrow{y^*}[k] \log y_k$$

$$\frac{\partial \mathcal{L}}{\partial \beta_{kj}} = \left(1 - \sigma\left(\beta_{y^*}^T h\right)\right) h_j$$

$$\frac{\partial \mathcal{L}}{\partial w_{jl}} = \left(1 - \sigma\left(\beta_{y^*}^T h\right)\right) \left(\beta_{y^* j} \sigma'\left(w_j^T x\right) x_l\right)$$

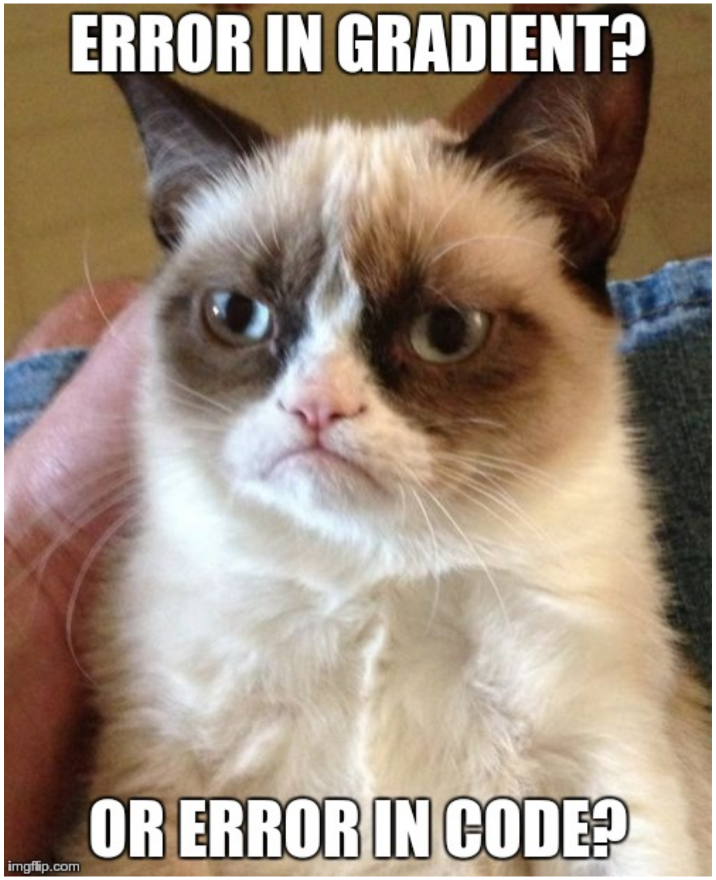
Debugging can be hard to do!

Gradients for Feed Forward Neural Network

$$y_k = \sigma\left(\beta_k^T \left(\sigma(w_j^T x + b_0)\right)_j\right)$$

h: a vector

$$\mathcal{L} = -\sum_k \overrightarrow{y^*}[k] \log y_k$$



$$\frac{\partial \mathcal{L}}{\partial \beta_{kj}} = \left(1 - \sigma(\beta_{y^*}^T h)\right) h_j$$

$$\frac{\partial \mathcal{L}}{\partial w_{jl}} = \left(1 - \sigma(\beta_{y^*}^T h)\right) \left(\beta_{y^* j} \sigma'(w_j^T x) x_l\right)$$

Debugging can be hard to do!

Outline

Neural networks: non-linear classifiers

Learning weights: backpropagation of error

Autodifferentiation (in reverse mode)

Finding Gradients

$$f(x_1, x_2) = x_1^2 + (x_1 - x_2)^a - \log(x_1^2 + x_2^2)$$

what are the partial derivatives?

Finding Gradients

$$f(x_1, x_2) = x_1^2 + (x_1 - x_2)^a - \log(x_1^2 + x_2^2)$$

$$\frac{\partial f(x_1, x_2)}{\partial x_1} = 2x_1 + a(x_1 - x_2)^{a-1} - \frac{2x_1}{x_1^2 + x_2^2}$$

Finding Gradients

$$f(x_1, x_2) = x_1^2 + (x_1 - x_2)^a - \log(x_1^2 + x_2^2)$$

⇓ *chain rule* (multiple times)

$$\frac{\partial f(x_1, x_2)}{\partial x_1} = 2x_1 + a(x_1 - x_2)^{a-1} - \frac{2x_1}{x_1^2 + x_2^2}$$

$$\frac{\partial f(x_1, x_2)}{\partial x_2} = -a(x_1 - x_2)^{a-1} - \frac{2x_2}{x_1^2 + x_2^2}$$

Autodifferentiation

$$f(x_1, x_2) = x_1^2 + (x_1 - x_2)^a - \log(x_1^2 + x_2^2)$$

$$z_1 = x_1^2$$

$$z_2 = x_2^2$$

$$z_3 = (x_1 - x_2)$$

$$z_4 = z_3^a$$

$$z_5 = z_1 + z_2$$

$$z_6 = \log z_5$$

$$z_7 = z_1 + z_4 - z_6$$

$$y = z_7$$

Autodifferentiation

$$f(x_1, x_2) = x_1^2 + (x_1 - x_2)^a - \log(x_1^2 + x_2^2)$$

$$z_1 = x_1^2$$
$$z_2 = x_2^2$$
$$z_3 = (x_1 - x_2)$$
$$z_4 = z_3^a$$
$$z_5 = z_1 + z_2$$
$$z_6 = \log z_5$$
$$z_7 = z_1 + z_4 - z_6$$
$$y = z_7$$

"straight line" program

autodiff: a way of finding gradients

mechanistic/procedural

two (standard) modes: forward and reverse

ML often uses reverse mode

Autodifferentiation

$$f(x_1, x_2) = x_1^2 + (x_1 - x_2)^a - \log(x_1^2 + x_2^2)$$

$z_1 = x_1^2$

$z_2 = x_2^2$

$z_3 = (x_1 - x_2)$

$z_4 = z_3^a$

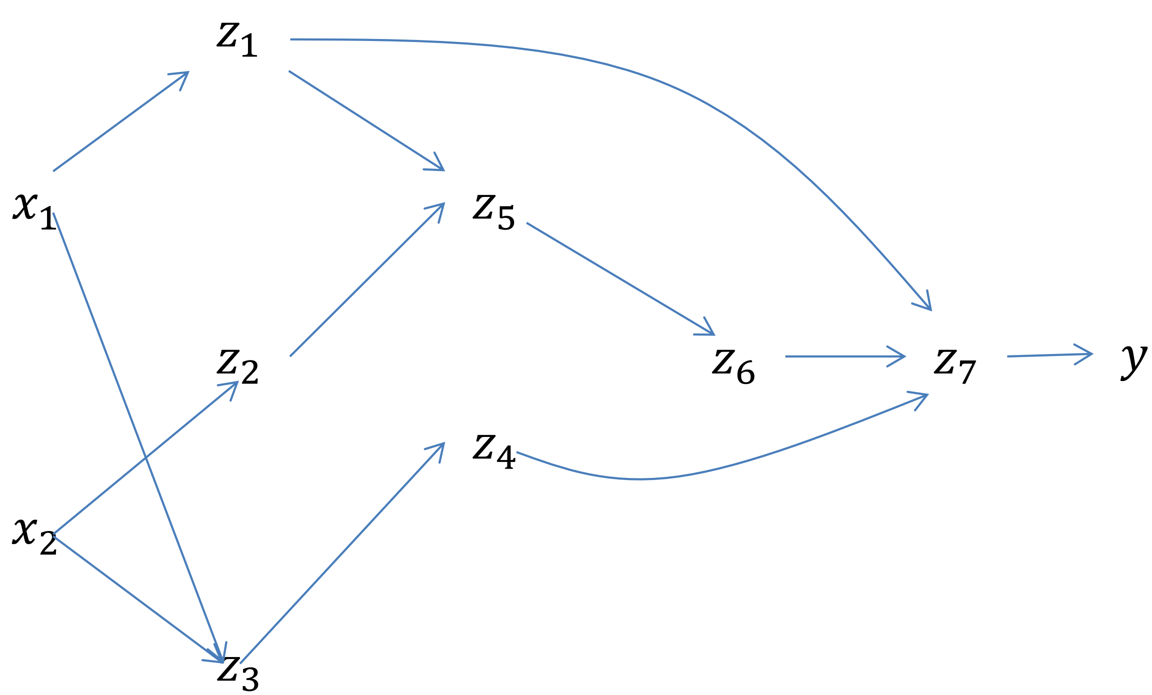
$z_5 = z_1 + z_2$

$z_6 = \log z_5$

$z_7 = z_1 + z_4 - z_6$

$y = z_7$

“straight line” program



computation graph

Autodifferentiation

$f(x_1, x_2) = x_1^2 + (x_1 - x_2)^a - \log(x_1^2 + x_2^2)$

$z_1 = x_1^2$

$z_2 = x_2^2$

$z_3 = (x_1 - x_2)$

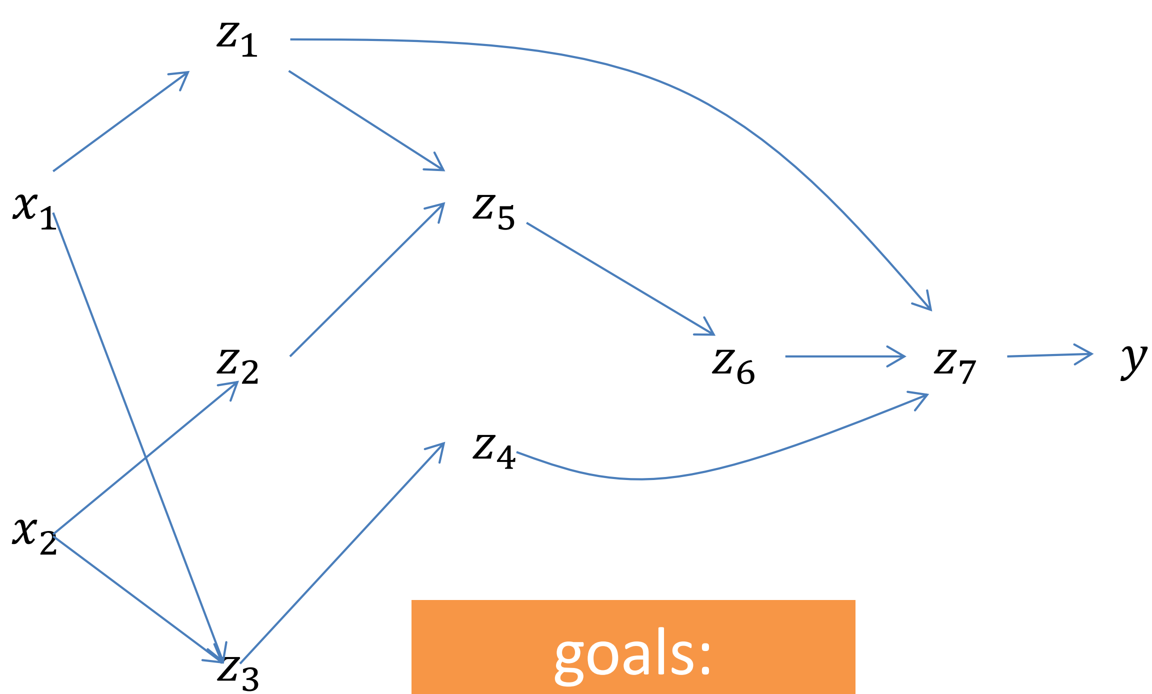
$z_4 = z_3^a$

$z_5 = z_1 + z_2$

$z_6 = \log z_5$

$z_7 = z_1 + z_4 - z_6$

$y = z_7$



"straight line" program

goals:

$\frac{\partial y}{\partial x_1}$ $\frac{\partial y}{\partial x_2}$

Autodifferentiation

$$f(x_1, x_2) = x_1^2 + (x_1 - x_2)^a - \log(x_1^2 + x_2^2)$$

$z_1 = x_1^2$

$z_2 = x_2^2$

$z_3 = (x_1 - x_2)$

$z_4 = z_3^a$

$z_5 = z_1 + z_2$

$z_6 = \log z_5$

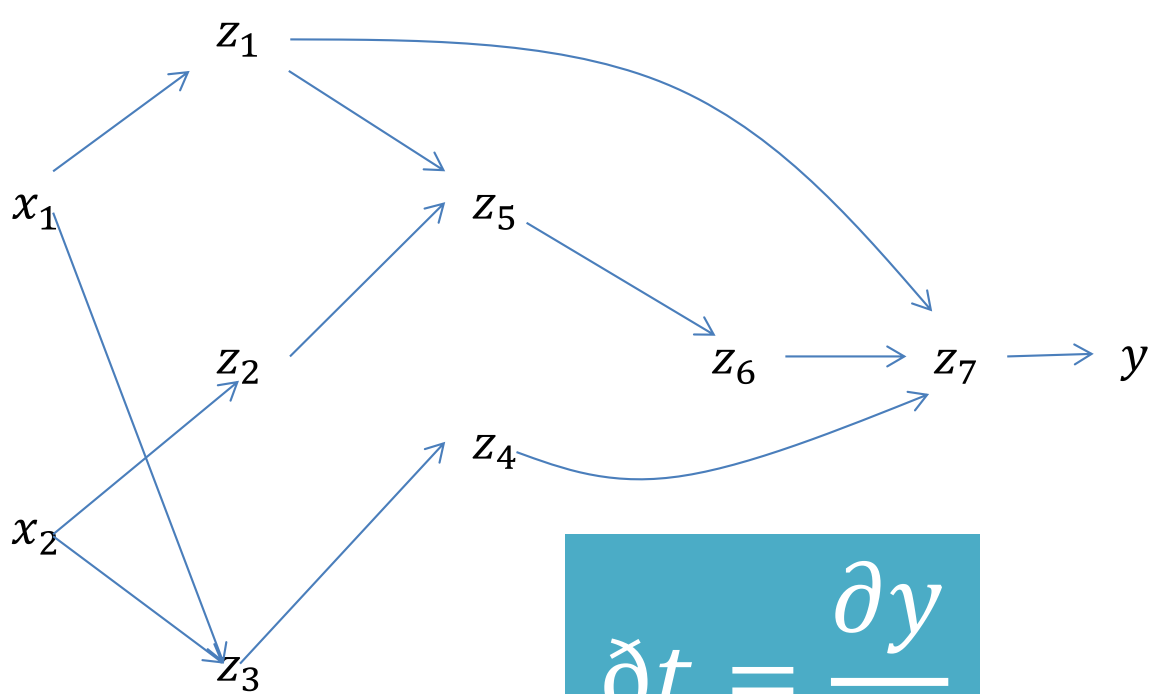
$z_7 = z_1 + z_4 - z_6$

$y = z_7$

goals:

$\frac{\partial y}{\partial x_1}$

$\frac{\partial y}{\partial x_2}$



ð$t = \frac{\partial y}{\partial t}$

adjoint

Autodifferentiation

$$ðt = \frac{\partial y}{\partial t}$$

adjoint

$$f(x_1, x_2) = x_1^2 + (x_1 - x_2)^a - \log(x_1^2 + x_2^2)$$

$z_1 = x_1^2$

$z_2 = x_2^2$

$z_3 = (x_1 - x_2)$

$z_4 = z_3^a$

$z_5 = z_1 + z_2$

$z_6 = \log z_5$

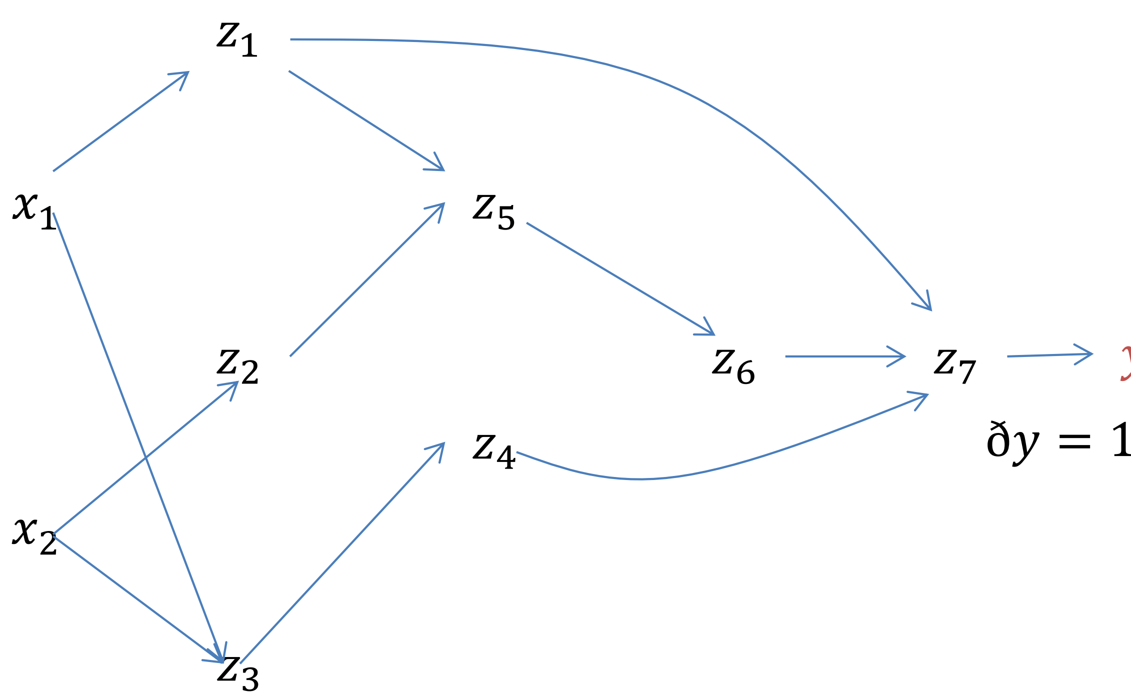
$z_7 = z_1 + z_4 - z_6$

$y = z_7$

goals:

$\frac{\partial y}{\partial x_1}$

$\frac{\partial y}{\partial x_2}$



Autodifferentiation

$$\eth t = \frac{\partial y}{\partial t}$$

adjoint

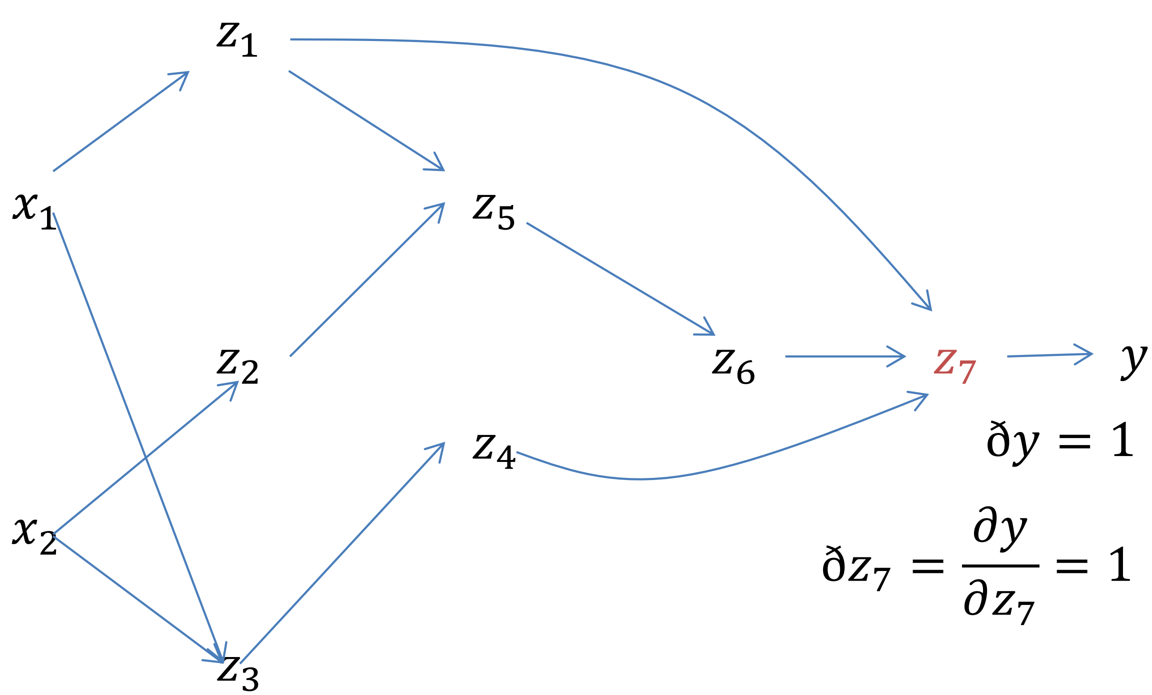
$$f(x_1, x_2) = x_1^2 + (x_1 - x_2)^a - \log(x_1^2 + x_2^2)$$

$$z_1 = x_1^2$$
$$z_2 = x_2^2$$
$$z_3 = (x_1 - x_2)$$
$$z_4 = z_3^a$$
$$z_5 = z_1 + z_2$$
$$z_6 = \log z_5$$
$$z_7 = z_1 + z_4 - z_6$$
$$y = z_7$$

goals:

$$\frac{\partial y}{\partial x_1}$$

$$\frac{\partial y}{\partial x_2}$$



$$\eth y = 1$$

$$\eth z_7 = \frac{\partial y}{\partial z_7} = 1$$

Autodifferentiation

$$\eth t = \frac{\partial y}{\partial t}$$

adjoint

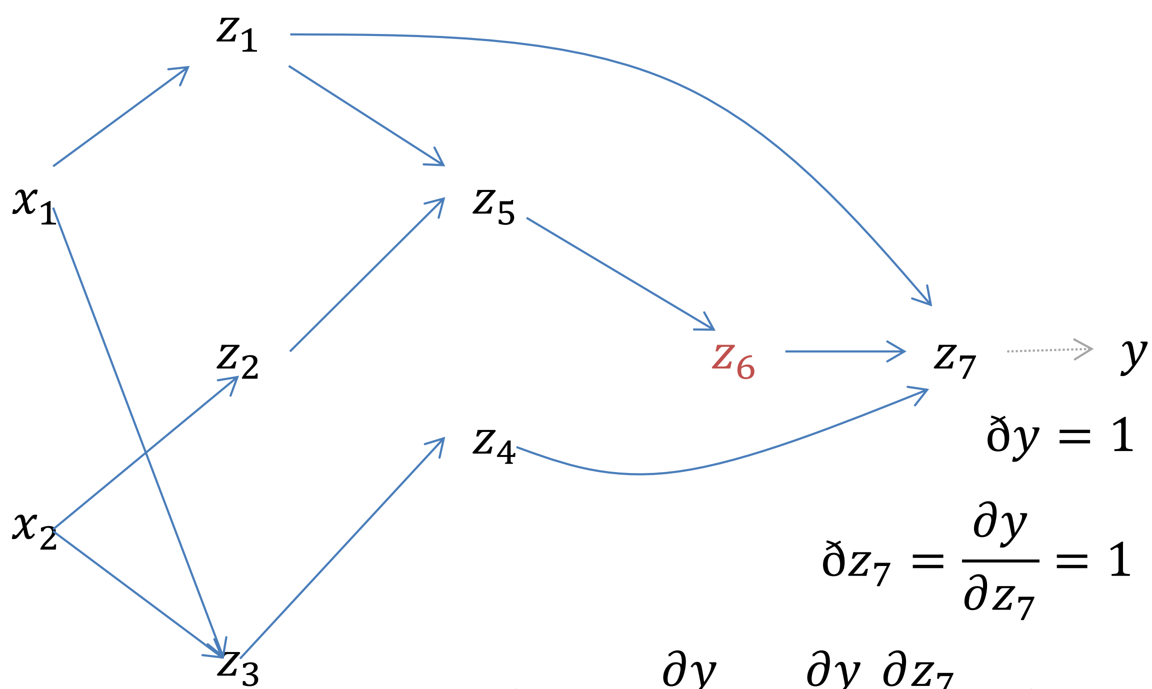
$$f(x_1, x_2) = x_1^2 + (x_1 - x_2)^a - \log(x_1^2 + x_2^2)$$

$$z_1 = x_1^2$$
$$z_2 = x_2^2$$
$$z_3 = (x_1 - x_2)$$
$$z_4 = z_3^a$$
$$z_5 = z_1 + z_2$$
$$z_6 = \log z_5$$
$$z_7 = z_1 + z_4 - z_6$$
$$y = z_7$$

goals:

$$\frac{\partial y}{\partial x_1}$$

$$\frac{\partial y}{\partial x_2}$$



$$\eth y = 1$$

$$\eth z_7 = \frac{\partial y}{\partial z_7} = 1$$

$$\eth z_6 = \frac{\partial y}{\partial z_6} = \frac{\partial y}{\partial z_7}\frac{\partial z_7}{\partial z_6} = \eth z_7 * -1$$

Autodifferentiation

$$\eth t = \frac{\partial y}{\partial t}$$

adjoint

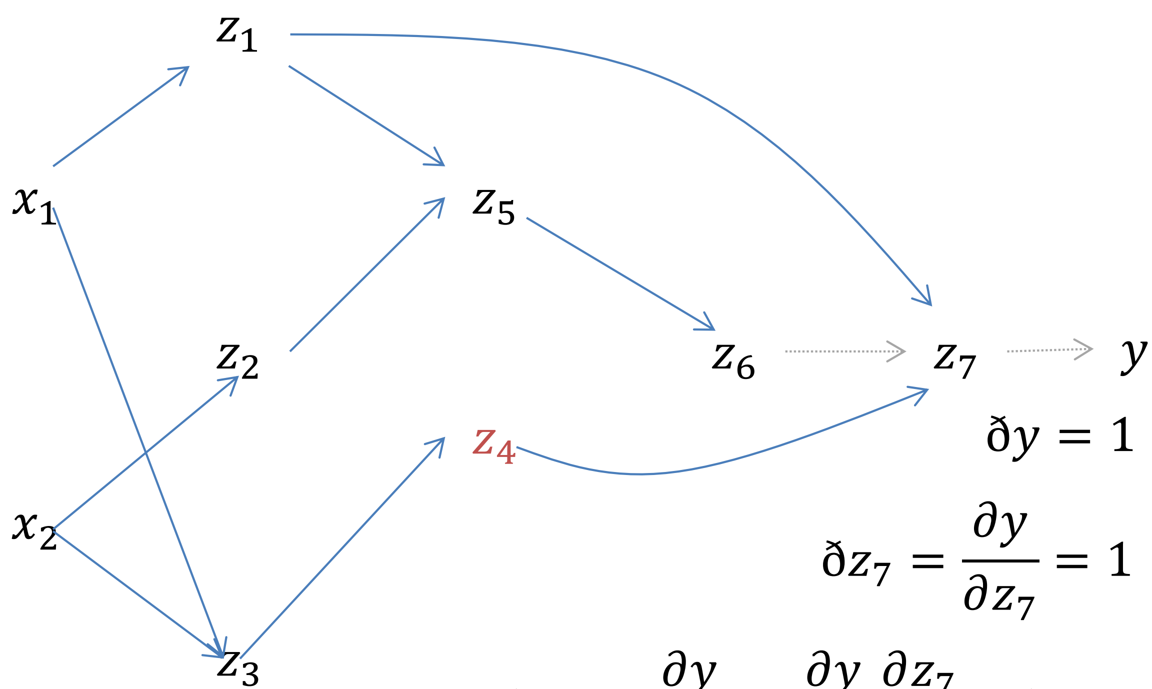
$$f(x_1, x_2) = x_1^2 + (x_1 - x_2)^a - \log(x_1^2 + x_2^2)$$

$$z_1 = x_1^2$$
$$z_2 = x_2^2$$
$$z_3 = (x_1 - x_2)$$
$$z_4 = z_3^a$$
$$z_5 = z_1 + z_2$$
$$z_6 = \log z_5$$
$$z_7 = z_1 + z_4 - z_6$$
$$y = z_7$$

goals:

$$\frac{\partial y}{\partial x_1}$$

$$\frac{\partial y}{\partial x_2}$$



$$\eth y = 1$$

$$\eth z_7 = \frac{\partial y}{\partial z_7} = 1$$

$$\eth z_6 = \frac{\partial y}{\partial z_6} = \frac{\partial y}{\partial z_7}\frac{\partial z_7}{\partial z_6} = \eth z_7 * -1$$

$$\eth z_4 = \frac{\partial y}{\partial z_4} = \frac{\partial y}{\partial z_7}\frac{\partial z_7}{\partial z_4} = \eth z_7 * 1$$

Autodifferentiation

$$\eth t = \frac{\partial y}{\partial t}$$

adjoint

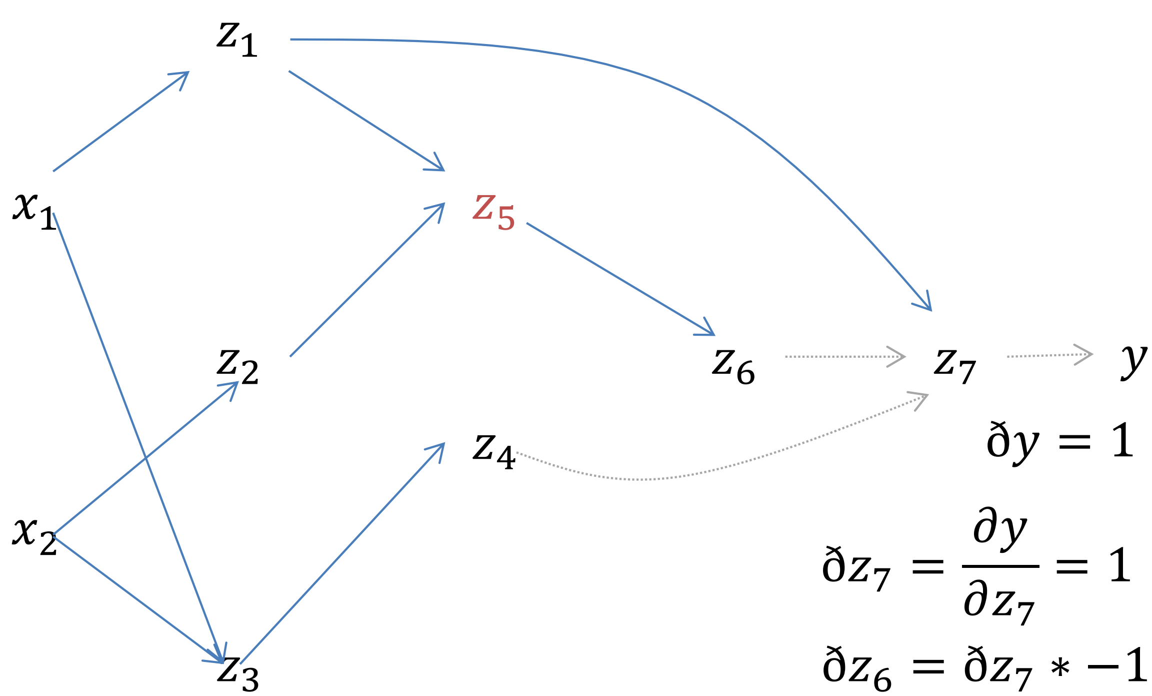
$$f(x_1, x_2) = x_1^2 + (x_1 - x_2)^a - \log(x_1^2 + x_2^2)$$

$$z_1 = x_1^2$$
$$z_2 = x_2^2$$
$$z_3 = (x_1 - x_2)$$
$$z_4 = z_3^a$$
$$z_5 = z_1 + z_2$$
$$z_6 = \log z_5$$
$$z_7 = z_1 + z_4 - z_6$$
$$y = z_7$$

goals:

$$\frac{\partial y}{\partial x_1}$$

$$\frac{\partial y}{\partial x_2}$$



$$\eth y = 1$$

$$\eth z_7 = \frac{\partial y}{\partial z_7} = 1$$

$$\eth z_6 = \eth z_7 * -1$$

$$\eth z_4 = \eth z_7 * 1$$

$$\eth z_5 = \frac{\partial y}{\partial z_5} = \frac{\partial y}{\partial z_7}\frac{\partial z_7}{\partial z_5} = \frac{\partial y}{\partial z_7}\frac{\partial z_7}{\partial z_6}\frac{\partial z_6}{\partial z_5}$$

Autodifferentiation

$$\eth t = \frac{\partial y}{\partial t}$$

adjoint

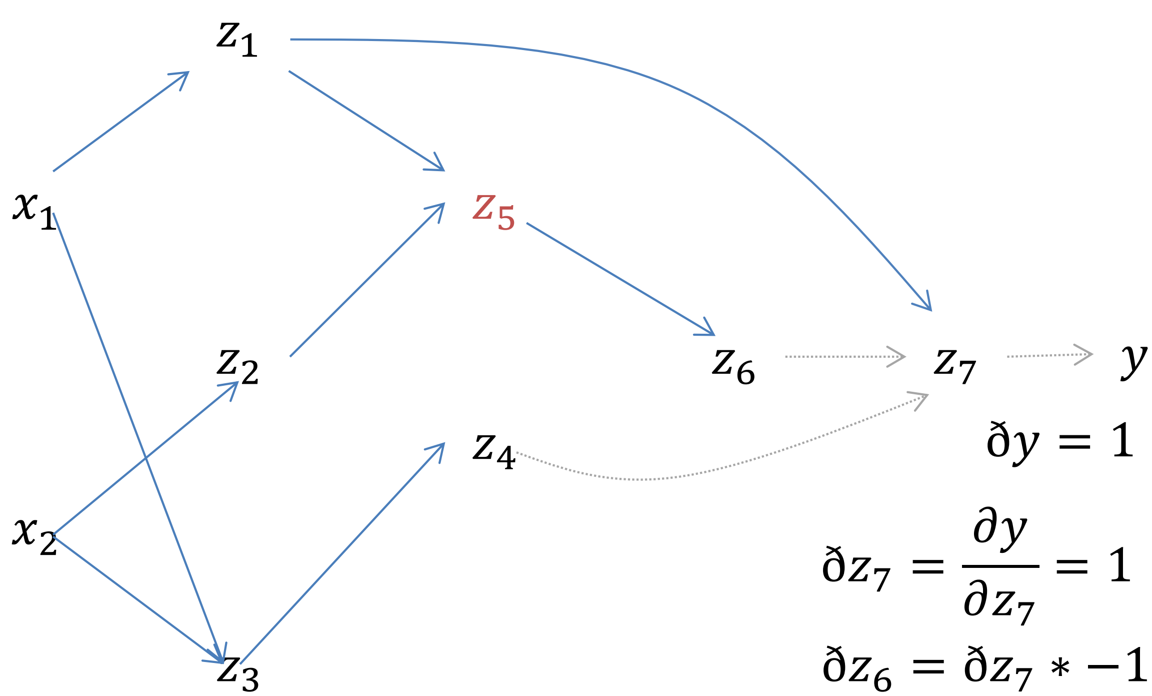
$$f(x_1, x_2) = x_1^2 + (x_1 - x_2)^a - \log(x_1^2 + x_2^2)$$

$$z_1 = x_1^2$$
$$z_2 = x_2^2$$
$$z_3 = (x_1 - x_2)$$
$$z_4 = z_3^a$$
$$z_5 = z_1 + z_2$$
$$z_6 = \log z_5$$
$$z_7 = z_1 + z_4 - z_6$$
$$y = z_7$$

goals:

$$\frac{\partial y}{\partial x_1}$$

$$\frac{\partial y}{\partial x_2}$$



$$\eth y = 1$$

$$\eth z_7 = \frac{\partial y}{\partial z_7} = 1$$

$$\eth z_6 = \eth z_7 * -1$$

$$\eth z_4 = \eth z_7 * 1$$

$$\eth z_5 = \frac{\partial y}{\partial z_5} = \frac{\partial y}{\partial z_7}\frac{\partial z_7}{\partial z_5} = \frac{\partial y}{\partial z_7}\frac{\partial z_7}{\partial z_6}\frac{\partial z_6}{\partial z_5} = \eth z_6 * \frac{1}{z_5}$$

Autodifferentiation

$$ðt = \frac{\partial y}{\partial t}$$

adjoint

$$f(x_1, x_2) = x_1^2 + (x_1 - x_2)^a - \log(x_1^2 + x_2^2)$$

$z_1 = x_1^2$

$z_2 = x_2^2$

$z_3 = (x_1 - x_2)$

$z_4 = z_3^a$

$z_5 = z_1 + z_2$

$z_6 = \log z_5$

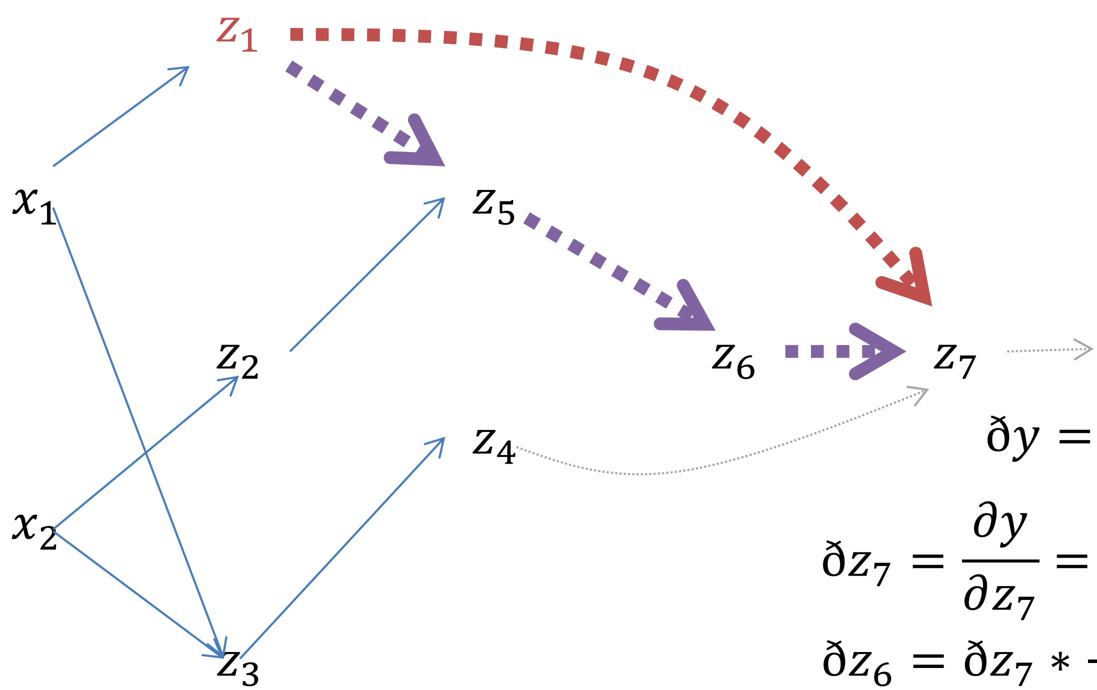
$z_7 = z_1 + z_4 - z_6$

$y = z_7$

goals:

$$\frac{\partial y}{\partial x_1}$$

$$\frac{\partial y}{\partial x_2}$$



$ðy = 1$

$$ðz_7 = \frac{\partial y}{\partial z_7} = 1$$

$ðz_6 = ðz_7 * -1$

$ðz_4 = ðz_7 * 1$

$$ðz_5 = ðz_6 * \frac{1}{z_5}$$

$$ðz_1 = \frac{\partial y}{\partial z_1} = \frac{\partial y}{\partial z_7}\frac{\partial z_7}{\partial z_1} + \frac{\partial y}{\partial z_7}\frac{\partial z_7}{\partial z_6}\frac{\partial z_6}{\partial z_5}\frac{\partial z_5}{\partial z_1}$$

Autodifferentiation

$$\eth t = \frac{\partial y}{\partial t}$$

adjoint

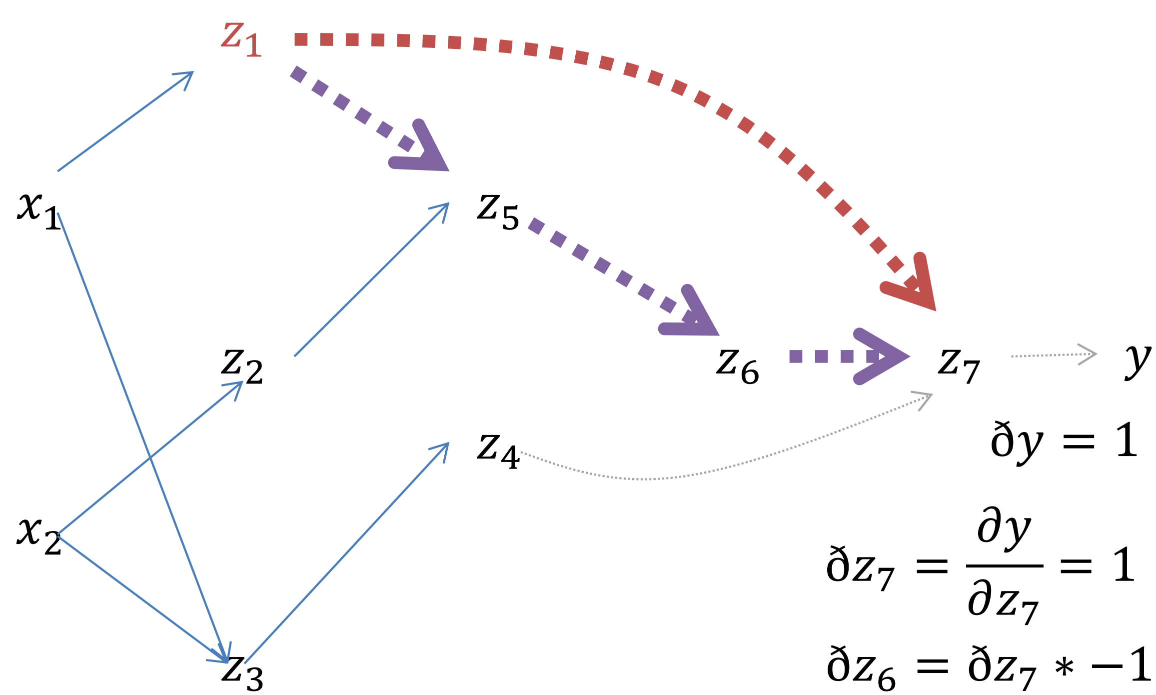
$$f(x_1, x_2) = x_1^2 + (x_1 - x_2)^a - \log(x_1^2 + x_2^2)$$

$$z_1 = x_1^2$$
$$z_2 = x_2^2$$
$$z_3 = (x_1 - x_2)$$
$$z_4 = z_3^a$$
$$z_5 = z_1 + z_2$$
$$z_6 = \log z_5$$
$$z_7 = z_1 + z_4 - z_6$$
$$y = z_7$$

goals:

$$\frac{\partial y}{\partial x_1}$$

$$\frac{\partial y}{\partial x_2}$$



$$\eth y = 1$$

$$\eth z_7 = \frac{\partial y}{\partial z_7} = 1$$

$$\eth z_6 = \eth z_7 * -1$$

$$\eth z_4 = \eth z_7 * 1$$

$$\eth z_5 = \eth z_6 * \frac{1}{z_5}$$

$$\eth z_1 = \frac{\partial y}{\partial z_1} = \eth z_7 * 1 + \eth z_5 * 1$$

Autodifferentiation

$$\eth t = \frac{\partial y}{\partial t}$$

adjoint

$$f(x_1, x_2) = x_1^2 + (x_1 - x_2)^a - \log(x_1^2 + x_2^2)$$

$z_1 = x_1^2$

$z_2 = x_2^2$

$z_3 = (x_1 - x_2)$

$z_4 = z_3^a$

$z_5 = z_1 + z_2$

$z_6 = \log z_5$

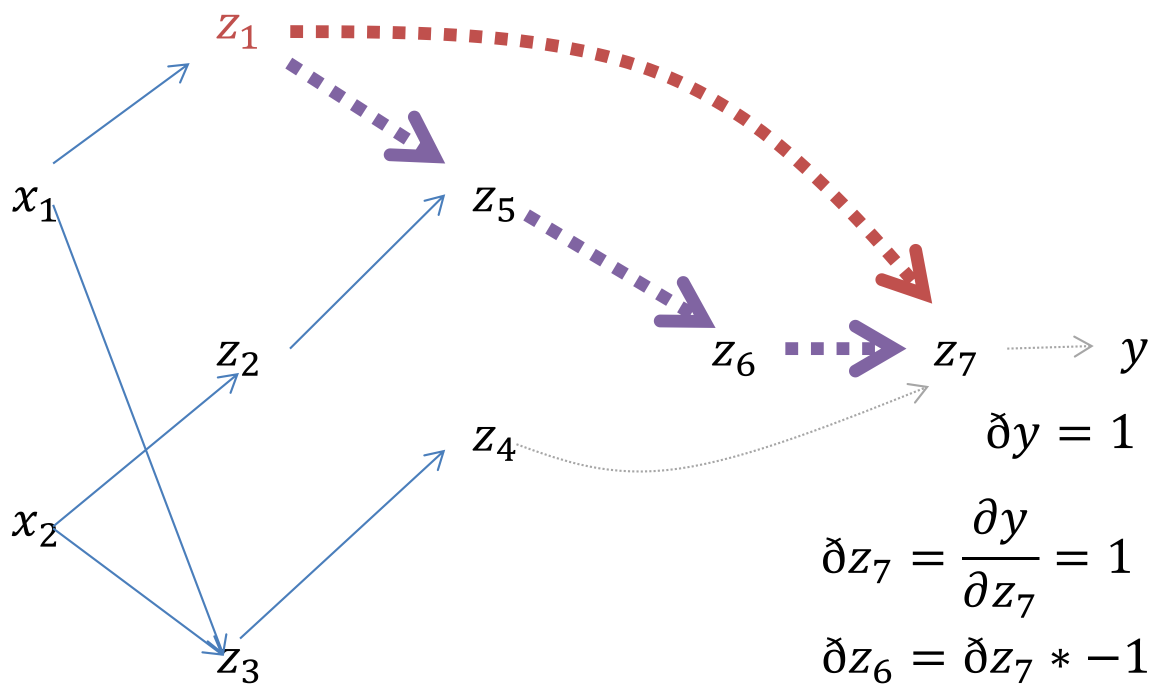
$z_7 = z_1 + z_4 - z_6$

$y = z_7$

goals:

$$\frac{\partial y}{\partial x_1}$$

$$\frac{\partial y}{\partial x_2}$$



$\eth y = 1$

$$\eth z_7 = \frac{\partial y}{\partial z_7} = 1$$

$\eth z_6 = \eth z_7 * -1$

$\eth z_4 = \eth z_7 * 1$

$$\eth z_5 = \eth z_6 * \frac{1}{z_5}$$

$\eth z_1 \mathrel{+}= \eth z_7 * 1$

$\eth z_1 \mathrel{+}= \eth z_5 * 1$

Autodifferentiation

$$\eth t = \frac{\partial y}{\partial t}$$

adjoint

$$f(x_1, x_2) = x_1^2 + (x_1 - x_2)^a - \log(x_1^2 + x_2^2)$$

$z_1 = x_1^2$

$z_2 = x_2^2$

$z_3 = (x_1 - x_2)$

$z_4 = z_3^a$

$z_5 = z_1 + z_2$

$z_6 = \log z_5$

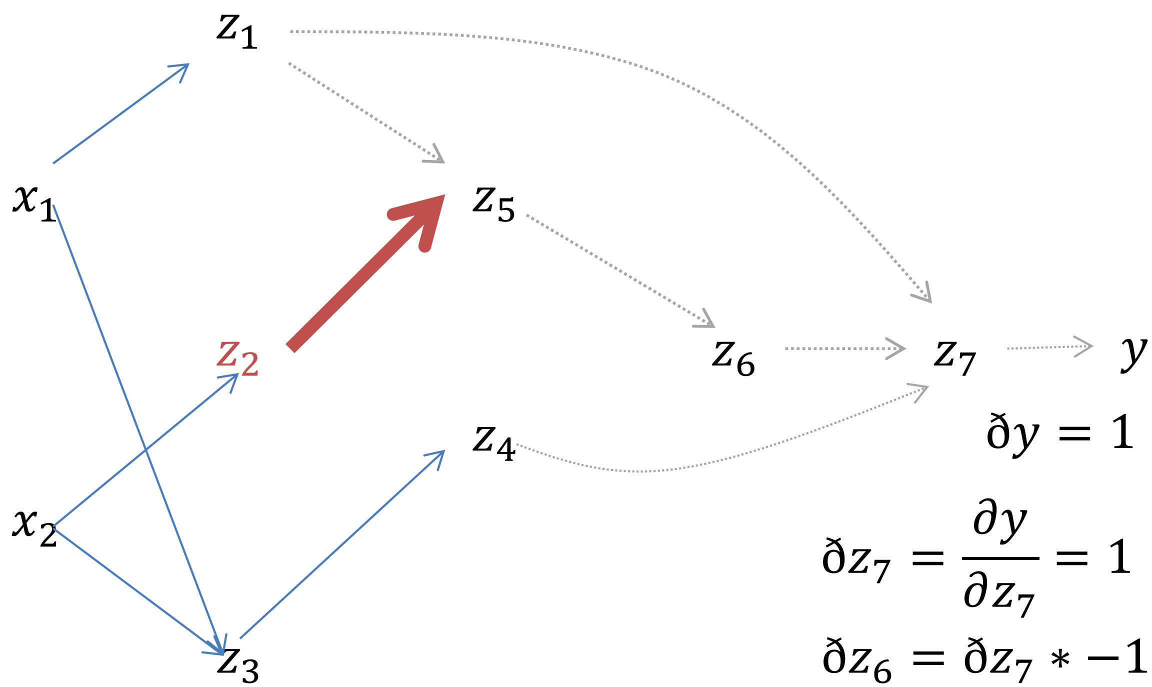
$z_7 = z_1 + z_4 - z_6$

$y = z_7$

goals:

$$\frac{\partial y}{\partial x_1}$$

$$\frac{\partial y}{\partial x_2}$$



$\eth y = 1$

$$\eth z_7 = \frac{\partial y}{\partial z_7} = 1$$

$\eth z_6 = \eth z_7 * -1$

$\eth z_4 = \eth z_7 * 1$

$$\eth z_5 = \eth z_6 * \frac{1}{z_5}$$

$\eth z_1 \mathrel{+}= \eth z_7 * 1$

$\eth z_1 \mathrel{+}= \eth z_5 * 1$

$$\eth z_2 = \frac{\partial y}{\partial z_2} = \eth z_5 * 1$$

Autodifferentiation

$$\eth t = \frac{\partial y}{\partial t}$$

adjoint

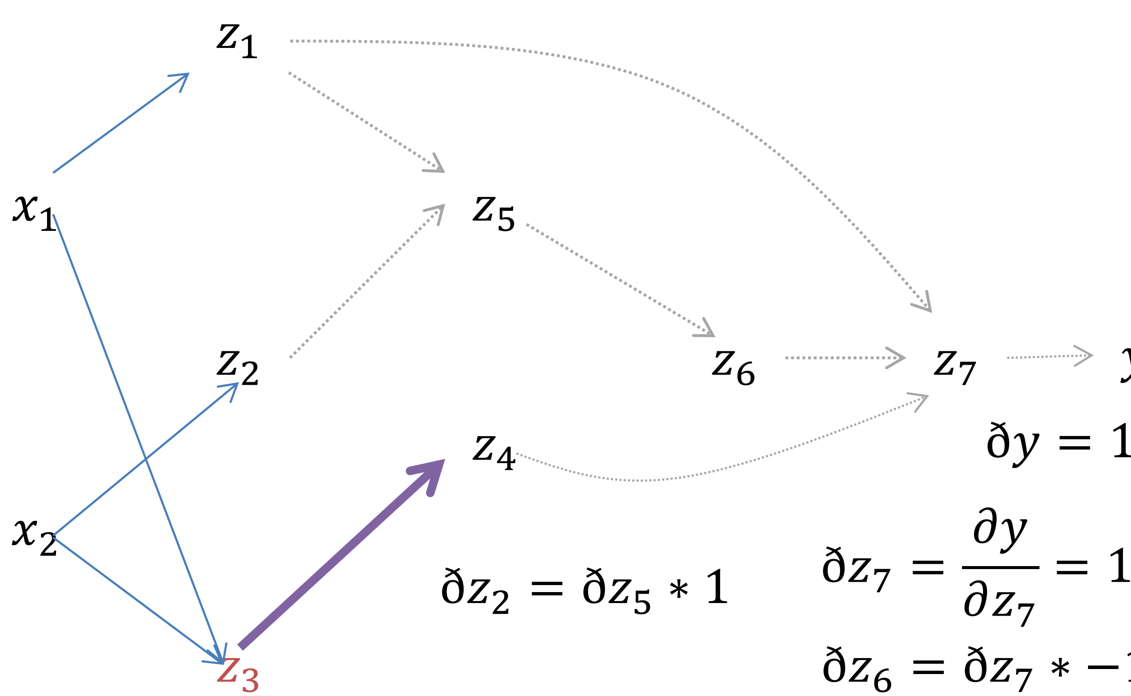
$$f(x_1, x_2) = x_1^2 + (x_1 - x_2)^a - \log(x_1^2 + x_2^2)$$

$$z_1 = x_1^2$$
$$z_2 = x_2^2$$
$$z_3 = (x_1 - x_2)$$
$$z_4 = z_3^a$$
$$z_5 = z_1 + z_2$$
$$z_6 = \log z_5$$
$$z_7 = z_1 + z_4 - z_6$$
$$y = z_7$$

goals:

$$\frac{\partial y}{\partial x_1}$$

$$\frac{\partial y}{\partial x_2}$$



$$\eth z_2 = \eth z_5 * 1$$

$$\eth y = 1$$

$$\eth z_7 = \frac{\partial y}{\partial z_7} = 1$$

$$\eth z_6 = \eth z_7 * -1$$

$$\eth z_4 = \eth z_7 * 1$$

$$\eth z_5 = \eth z_6 * \frac{1}{z_5}$$

$$\eth z_1 \mathrel{+}= \eth z_7 * 1$$

$$\eth z_1 \mathrel{+}= \eth z_5 * 1$$

$$\eth z_3 = \frac{\partial y}{\partial 3} = \eth z_4 * a * z_3^{a-1}$$

Autodifferentiation

$$\eth t = \frac{\partial y}{\partial t}$$

adjoint

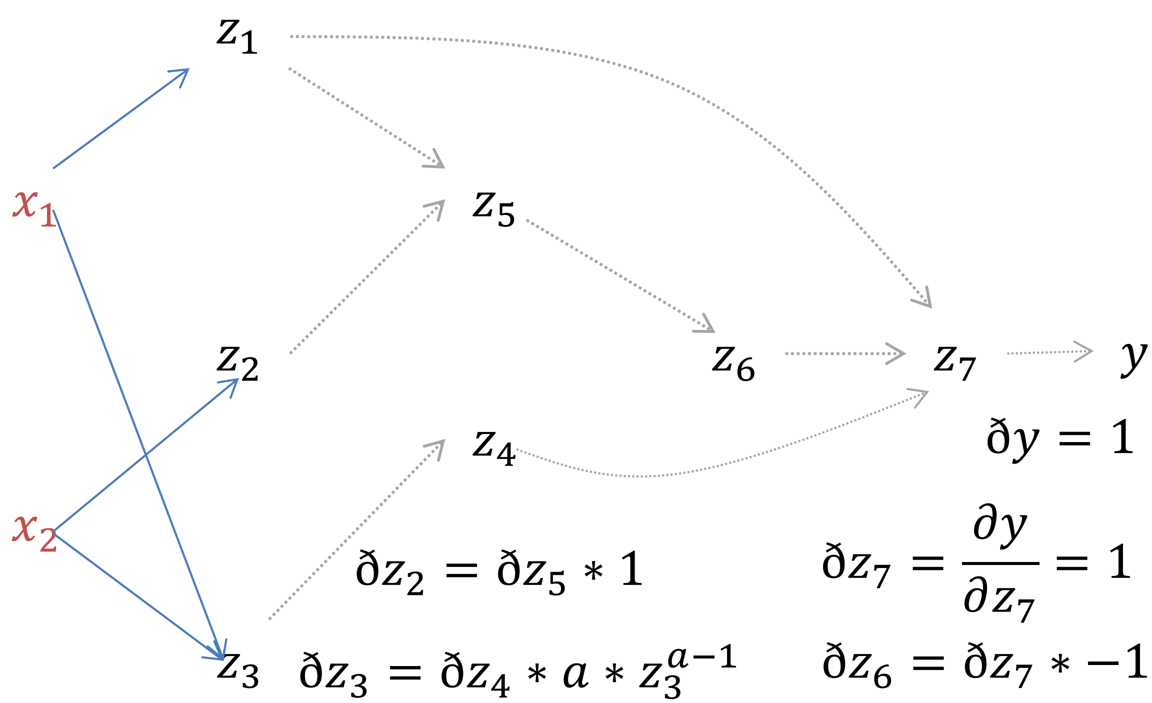
$$f(x_1, x_2) = x_1^2 + (x_1 - x_2)^a - \log(x_1^2 + x_2^2)$$

$$z_1 = x_1^2$$
$$z_2 = x_2^2$$
$$z_3 = (x_1 - x_2)$$
$$z_4 = z_3^a$$
$$z_5 = z_1 + z_2$$
$$z_6 = \log z_5$$
$$z_7 = z_1 + z_4 - z_6$$
$$y = z_7$$

goals:

$$\frac{\partial y}{\partial x_1}$$

$$\frac{\partial y}{\partial x_2}$$



$$\eth y = 1$$

$$\eth z_7 = \frac{\partial y}{\partial z_7} = 1$$

$$\eth z_6 = \eth z_7 * -1$$

$$\eth z_4 = \eth z_7 * 1$$

$$\eth z_5 = \eth z_6 * \frac{1}{z_5}$$

$$\eth z_1 \mathrel{+}= \eth z_7 * 1$$

$$\eth z_1 \mathrel{+}= \eth z_5 * 1$$

$$\eth z_2 = \eth z_5 * 1$$

$$\eth z_3 = \eth z_4 * a * z_3^{a-1}$$

$$\eth x_1 \mathrel{+}= \eth z_1 * 2x_1$$

$$\eth x_1 \mathrel{+}= \eth z_3 * 1$$

$$\eth x_2 \mathrel{+}= \eth z_2 * 2x_2$$

$$\eth x_2 \mathrel{+}= \eth z_3 * -1$$

Autodifferentiation

$$\eth t = \frac{\partial y}{\partial t}$$

adjoint

$$f(x_1, x_2) = x_1^2 + (x_1 - x_2)^a - \log(x_1^2 + x_2^2)$$

$z_1 = x_1^2$

$z_2 = x_2^2$

$z_3 = (x_1 - x_2)$

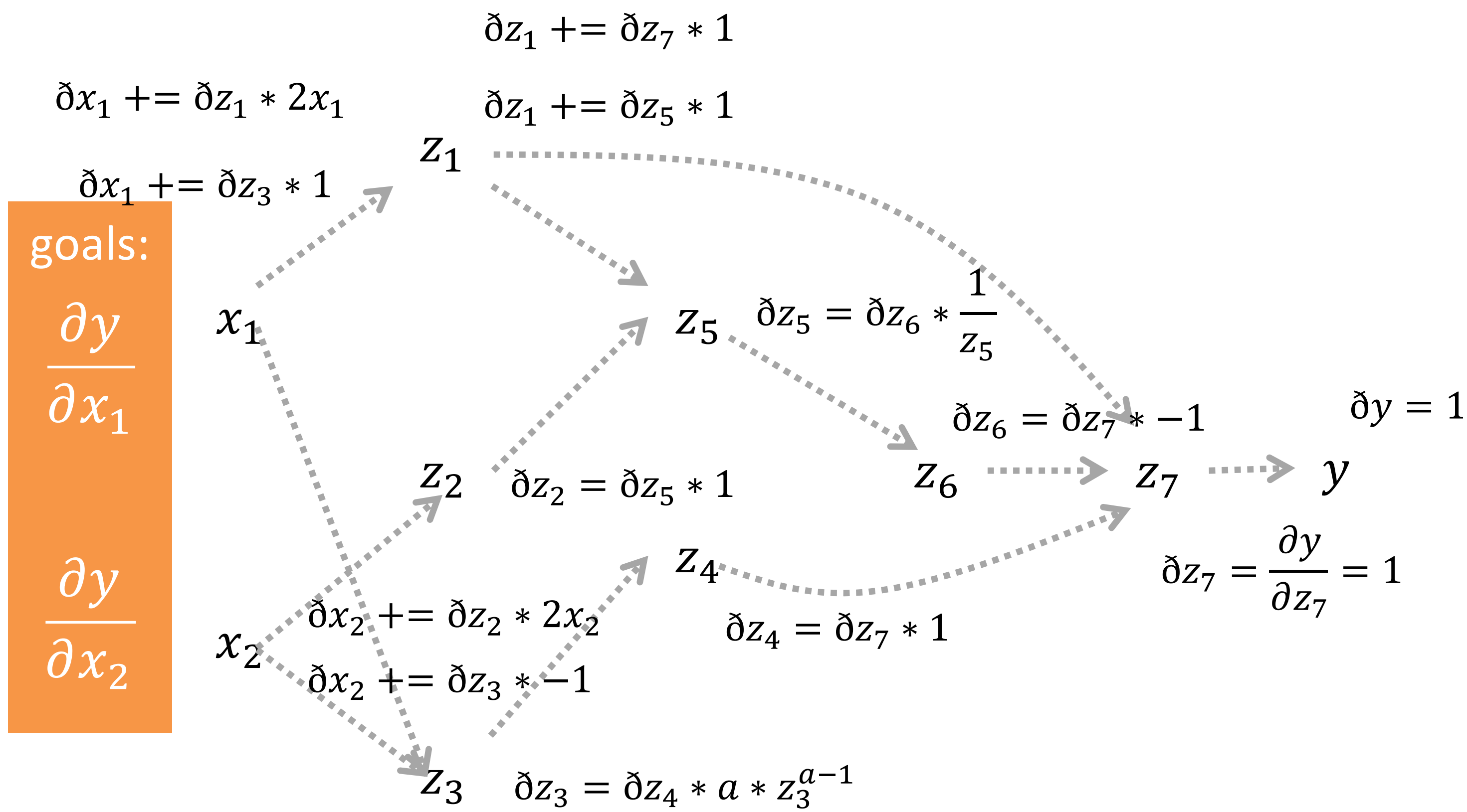
$z_4 = z_3^a$

$z_5 = z_1 + z_2$

$z_6 = \log z_5$

$z_7 = z_1 + z_4 - z_6$

$y = z_7$



Autodifferentiation

$$ðt = \frac{\partial y}{\partial t}$$

adjoint

$$f(x_1, x_2) = x_1^2 + (x_1 - x_2)^a - \log(x_1^2 + x_2^2)$$

$z_1 = x_1^2$

$z_2 = x_2^2$

$z_3 = (x_1 - x_2)$

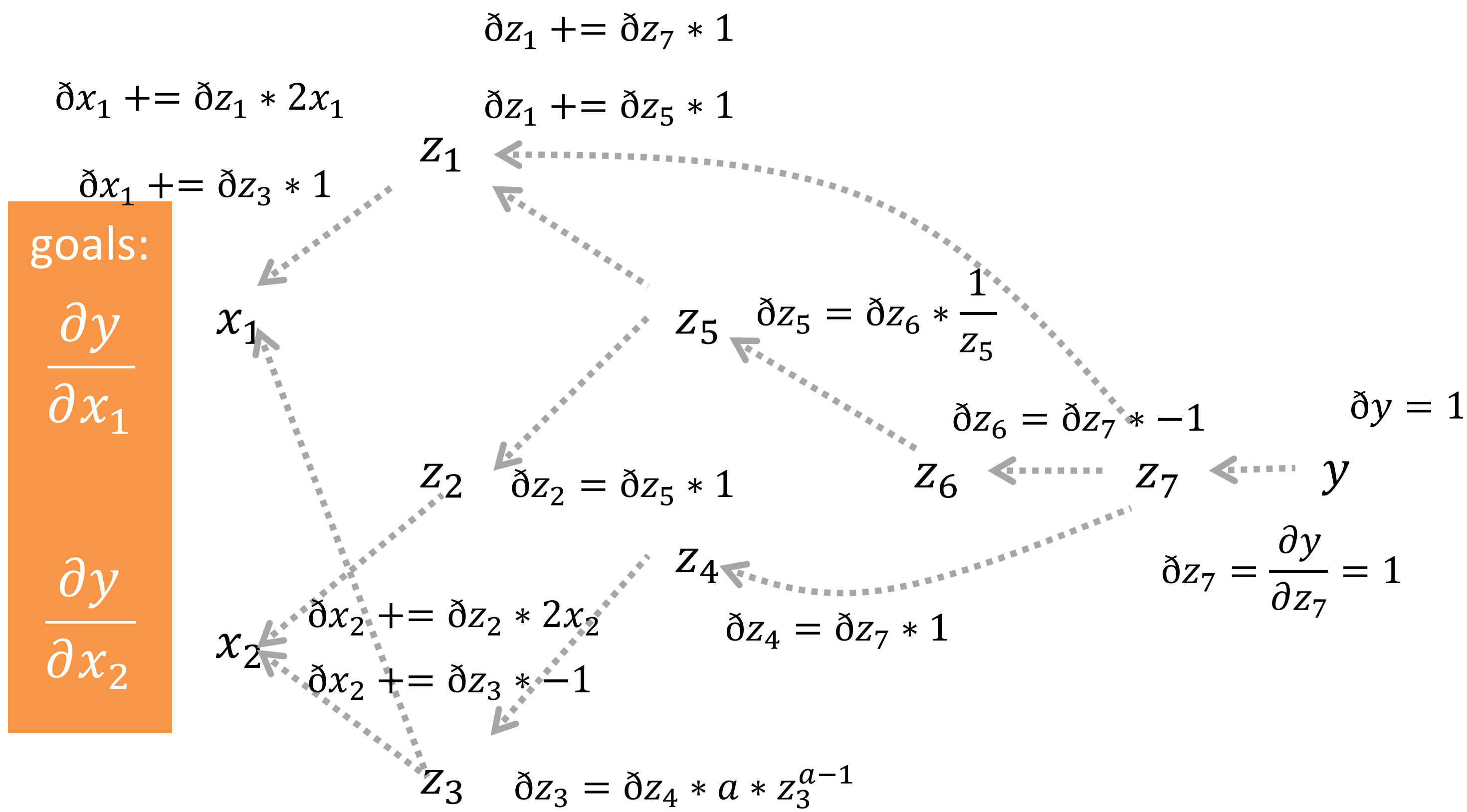
$z_4 = z_3^a$

$z_5 = z_1 + z_2$

$z_6 = \log z_5$

$z_7 = z_1 + z_4 - z_6$

$y = z_7$



autodifferentiation in reverse mode

Autodifferentiation in Reverse Mode

$$f(x_1, x_2) = x_1^2 + (x_1 - x_2)^a - \log(x_1^2 + x_2^2)$$

$$ðt = \frac{\partial y}{\partial t}$$

adjoint

$z_1 = x_1^2$

$z_2 = x_2^2$

$z_3 = (x_1 - x_2)$

$z_4 = z_3^a$

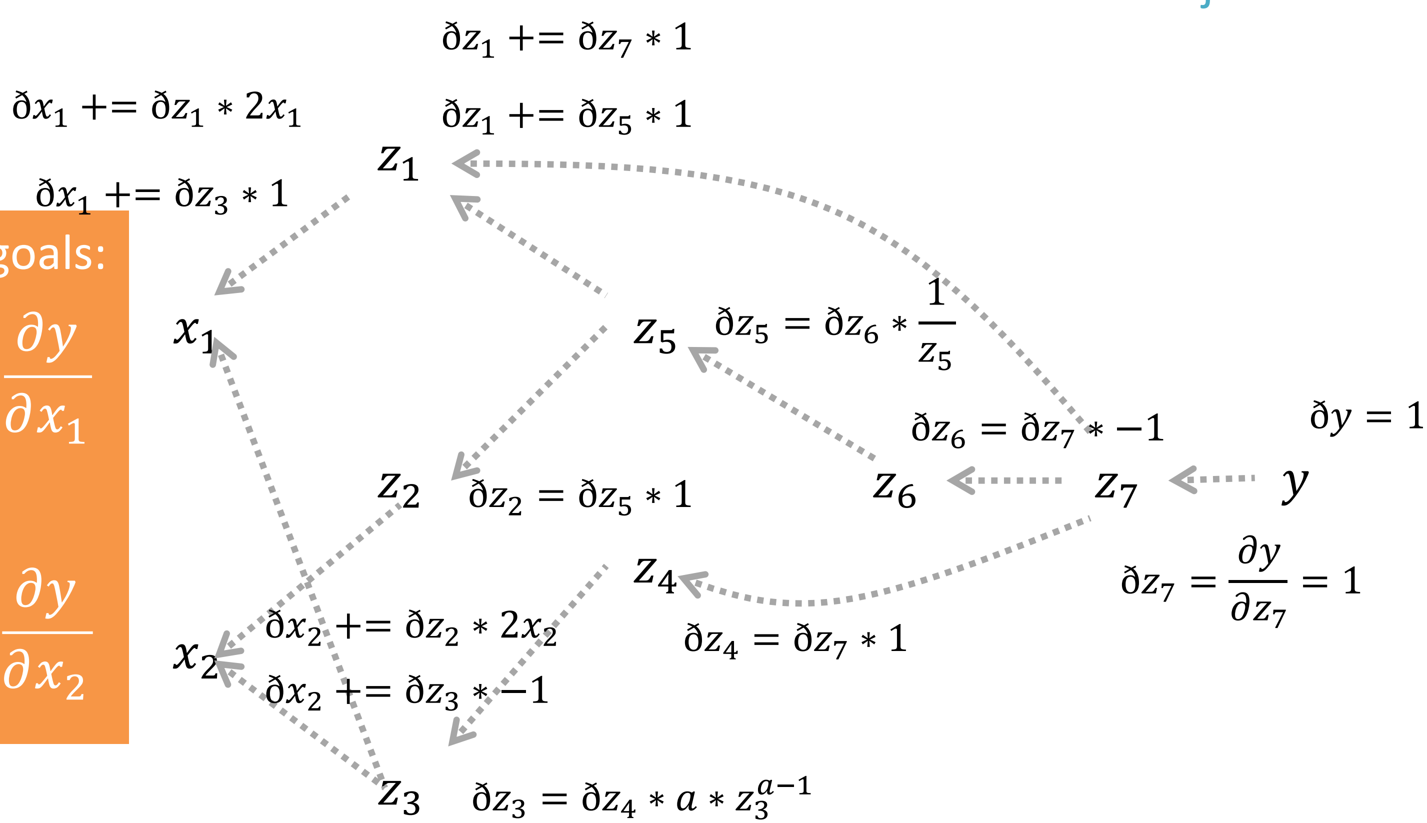
$z_5 = z_1 + z_2$

$z_6 = \log z_5$

$z_7 = z_1 + z_4 - z_6$

$y = z_7$

goals: $\frac{\partial y}{\partial x_1}$ $\frac{\partial y}{\partial x_2}$



$x_1 = 2$
$x_2 = 1$
$a = 1$

$f(x_1 = 2, x_2 = 1) \approx 3.390562$

$\nabla_x = (4.2, -1.4)$ *by exact gradients*

$\nabla_x = (4.2, -1.4)$ *by autodiff*

Code Proof of Autodiff

```
>> def f(x1, x2):
        return x1**2 + (x1-x2)**1 -
numpy.log(x1**2+x2**2)
```

```
>> def autodiff(x1,x2,a=1.0):
        z1=x1**2
        z2=x2**2
        z3=(x1-x2)
        z4=z3**a
        z5=z1+z2
        z6=numpy.log(z5)
        z7=z1+z4-z6
        y=z7
        dy=1
        dz7=dy
        dz6=dz7*-1.0
        dz5=dz6*1.0/z5
        dz4=dz7*1.0
        dz3=dz4*a*z3**(a-1)
        dz2=dz5*1.0
        dz1=dz7*1.0 +dz5*1.0
        dx1=dz1*2*x1+dz3*1.0
        dx2=dz2*2*x2+dz3*-1.0
        return dx1, dx2

>> autodiff(2,1)
(4.2, -1.4)
```

Code Proof of Autodiff

```
>> def f(x1, x2):
        return x1**2 + (x1-x2)**1 -
numpy.log(x1**2+x2**2)
```

forward pass

backward pass

```
>> def autodiff(x1,x2,a=1.0):
        z1=x1**2
        z2=x2**2
        z3=(x1-x2)
        z4=z3**a
        z5=z1+z2
        z6=numpy.log(z5)
        z7=z1+z4-z6
        y=z7
        dy=1
        dz7=dy
        dz6=dz7*-1.0
        dz5=dz6*1.0/z5
        dz4=dz7*1.0
        dz3=dz4*a*z3**(a-1)
        dz2=dz5*1.0
        dz1=dz7*1.0 +dz5*1.0
        dx1=dz1*2*x1+dz3*1.0
        dx2=dz2*2*x2+dz3*-1.0
        return dx1, dx2
```

```
>> autodiff(2,1)
(4.2, -1.4)
```

Outline

Neural networks: non-linear classifiers

Learning weights: backpropagation of error

Autodifferentiation (in reverse mode)

Gradient Descent:
Backpropagate the Error

Set t = 0
Pick a starting value θ_t
Until converged:
  for example(s) i:
1. Compute loss l on x_i
2. Get gradient $g_t = l'(x_i)$
3. Get scaling factor ρ_t
4. Set $\theta_{t+1} = \theta_t - \rho_t * g_t$
5. Set t += 1