Self-Supervised Learning Feed-Forward Nets

CSCI 601 471/671
NLP: Self-Supervised Models

https://self-supervised.cs.jhu.edu/sp2023/

[Slide credit: Andrej Karpathy, Fei Fei Li, Chris Manning and many others]

HW2 is released

- Did you see it?
- Due Tuesday noon, in 120 hours!

Recap: Contextual Representation of Meaning

- Language is complex, and ***context*** can completely change the meaning of a word in a sentence.
- Example:
  - I let the kids outside to ***play***.
  - He had never acted in a more famous ***play*** before.
  - It wasn't a ***play*** the coach would approve of.
- Previous models (e.g., Word2Vec) only have one representation per word
  - They can't capture these ambiguities.
- Need a model which captures the different nuances of the meaning of words given the surrounding text.
- **Approach:** build representations that are **contextual** via **neural networks**.

Chapter Plan

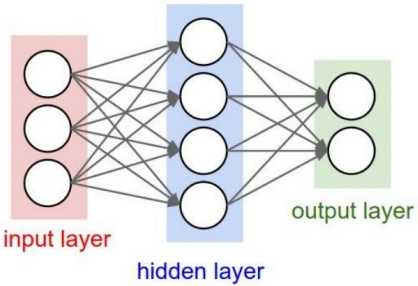
1. Defining neural networks (feed-forward nets)
2. Neural nets: brief history
3. Word2Vec as a simple neural network
4. Training neural networks: analytical back-propagation
5. Backprop in practice

Chapter goal: Get really comfortable with thinking, designing and building neural networks — very powerful modeling tools.

Neural Network

- Neural Networks are functions!
  - Function class for approximating real-valued, discrete-valued and vector valued target functions.
  - NN: $\boldsymbol{X} \rightarrow \boldsymbol{Y}$ where $\boldsymbol{X} = [0,1]^n$, or $\mathbb{R}^n$ and $\boldsymbol{Y} = [0,1]^d, \{0,1\}^d$

- Example: A **2-layer** neural network
  - The input, hidden and output variables are represented by nodes
  - The links are the weight parameters
  - Arrows denote direction of information flow through the network



$$f(\mathbf{x}) = W_2\, g(W_1 \mathbf{x}) \qquad \mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^d$$

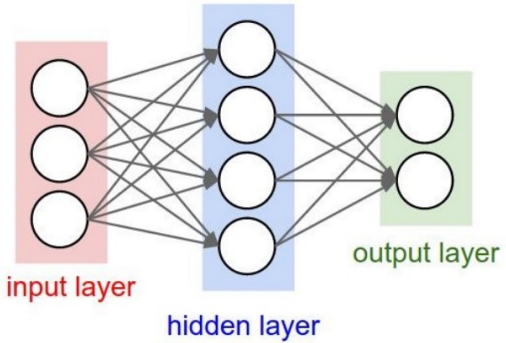
$$g(\mathbf{z}) = [\sigma(z_1), \ldots, \sigma(z_h)] \text{ (nonlinearity)} \qquad \sigma(z_i) = \frac{1}{1+e^{-x}} \text{ (sigmoid function)}$$

  - $W_1 \in \mathbb{R}^{h \times n}$ and $W_2 \in \mathbb{R}^{d \times h}$ are the parameters that need to be learned.

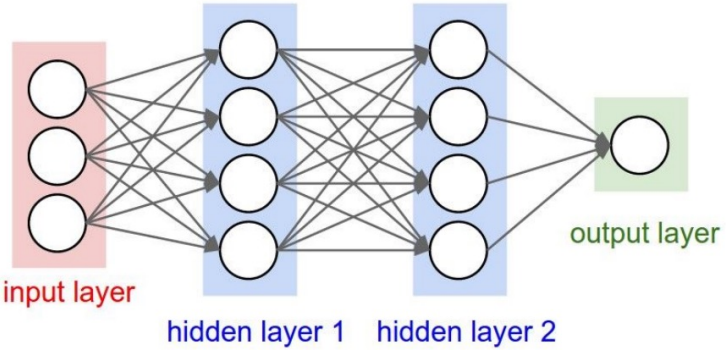
Neural Network: Making it bigger

Add more layers, or wider layers!

A **2-layer** neural network

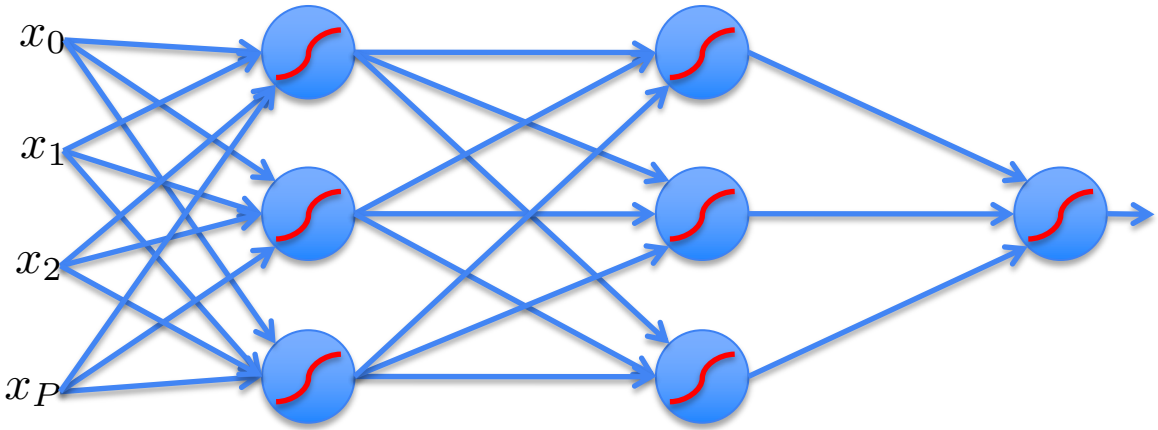


A **3-layer** neural network



Feed-forward Networks

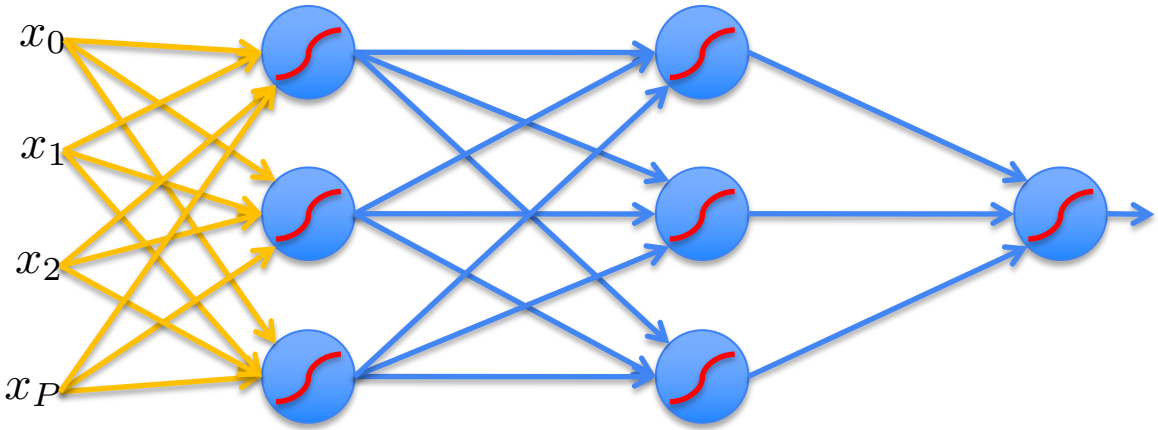
- This is actually a particular class called "feed-forward" networks.
  - Cascade neurons together
  - Output from one layer is the input to the next
  - Each layer has its own sets of weights



[Slides: HKUST]

Feed-forward networks

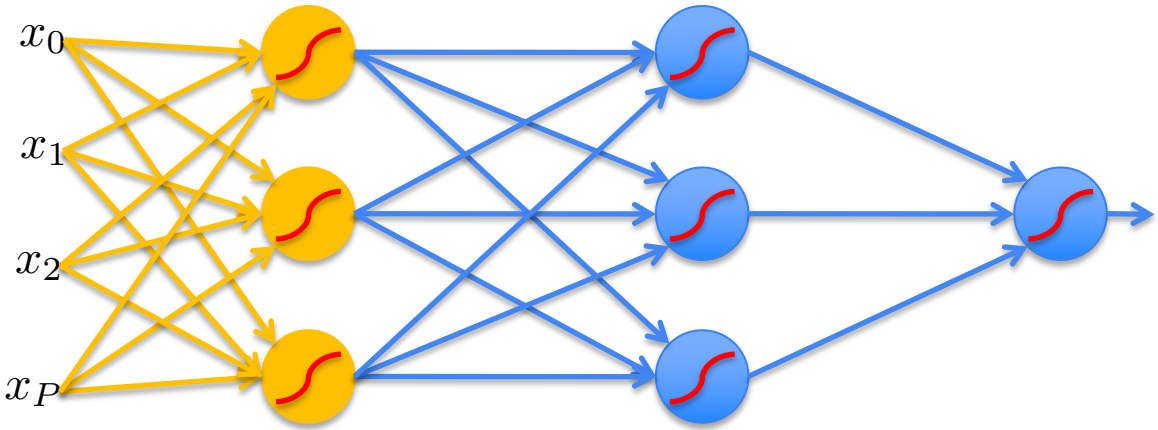
- Inputs multiplied by initial set of weights



[Slide: HKUST]

Feed-forward networks

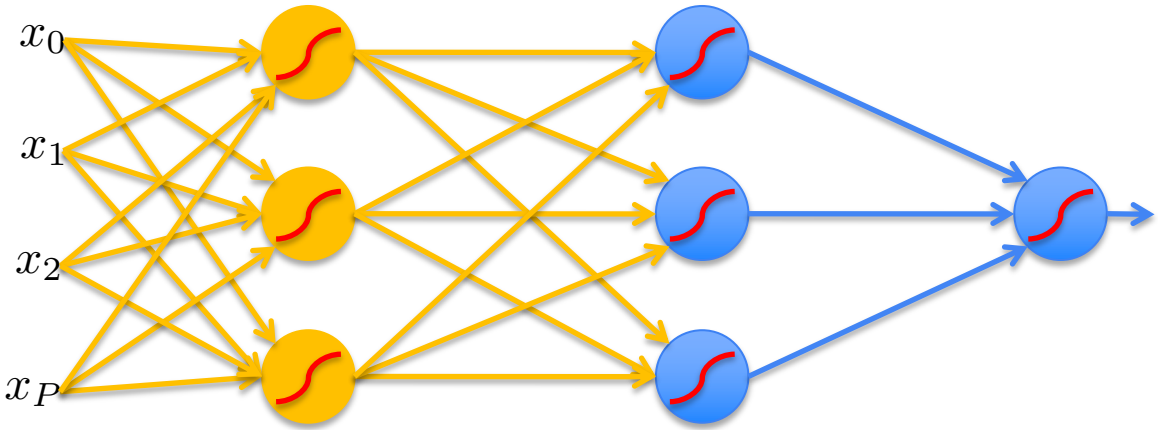
- Intermediate "predictions" computed at first hidden layer



[Slide: HKUST]

Feed-forward networks

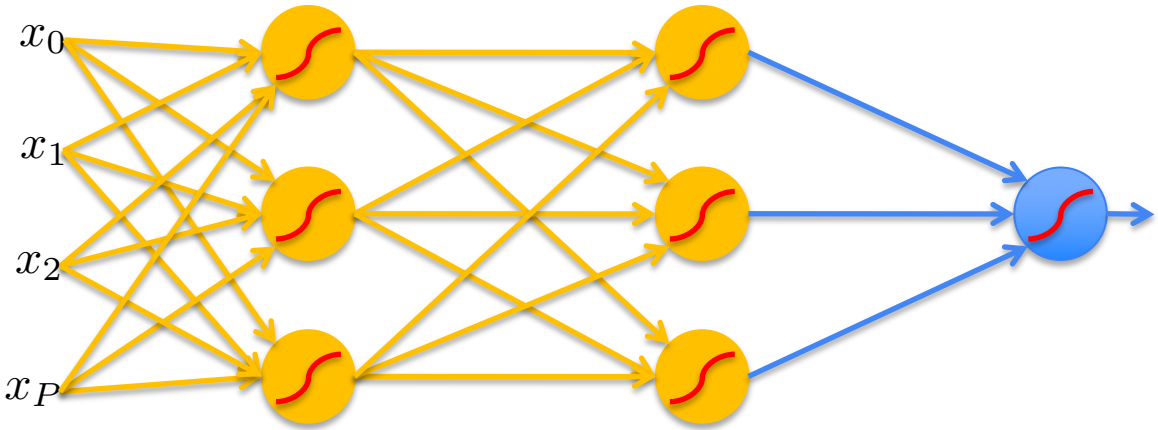
- Intermediate predictions multiplied by second layer of weights
- Predictions are fed forward through the network



[Slide: HKUST]

Feed-forward networks

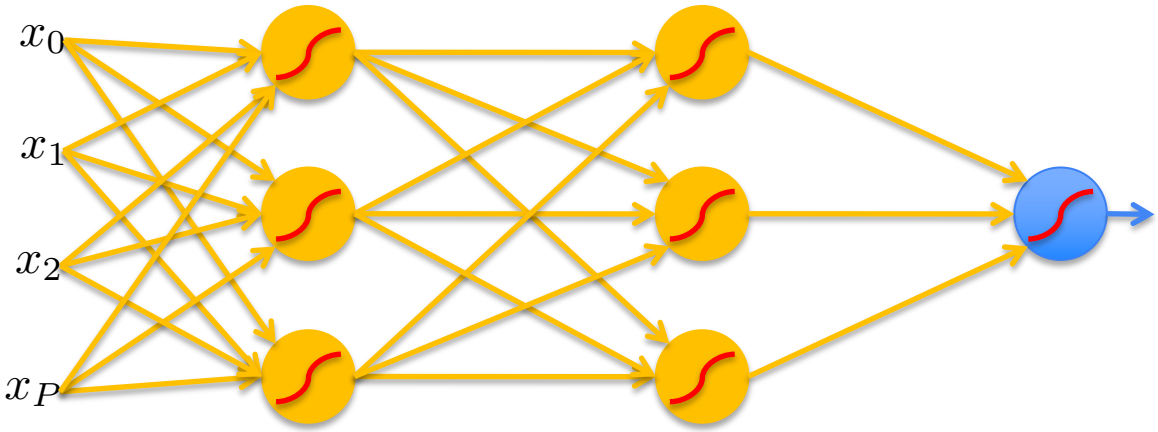
- Compute second set of intermediate predictions



[Slide: HKUST]

Feed-forward networks

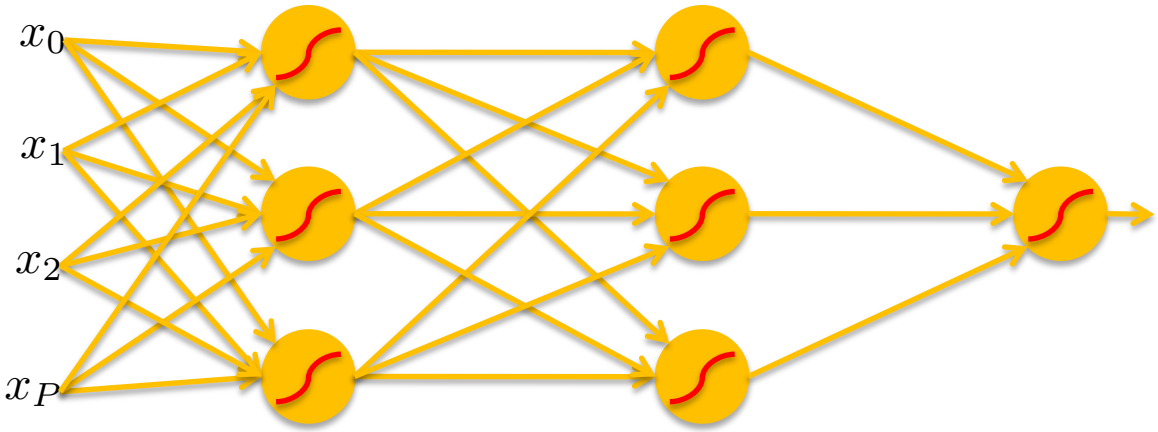
- Multiply by final set of weights



[Slide: HKUST]

Feed-forward networks

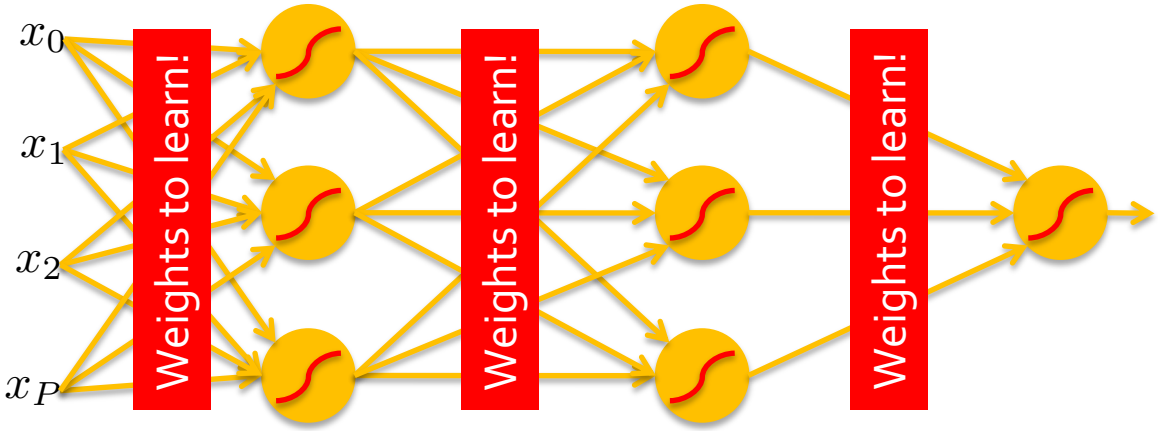
- Aggregate all the computations in the output
  - e.g. probability of a particular class



[Slide: HKUST]

Feed-forward networks

- All the intermediate parameters are ought to be learned.



[Slide: HKUST]

Why Add Non-linearity?

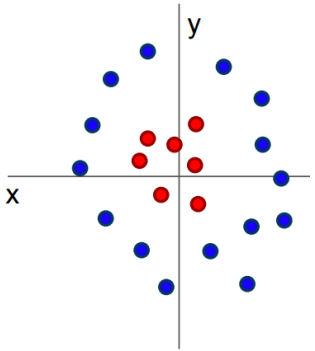
- Without non-linearity, the overall model amounts to a linear model.

$$f(\mathbf{x}) = W_2\, g(W_1\mathbf{x}) \qquad \tilde{f}(\mathbf{x}) = W_2W_1\mathbf{x} = W_3\mathbf{x}$$ (a linear function)

drop g

- A linear function cannot approximate complex tasks.

- Non-linearity adds capacity to the model to approximate any continuous function to arbitrary accuracy given sufficiently many hidden units.
  - See "universal approximation theorem"

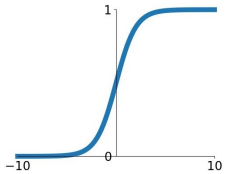


Cannot separate red and blue points with linear classifier

Activation/Nonlinearity Functions

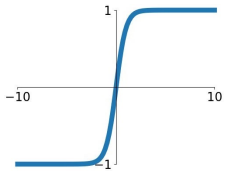
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



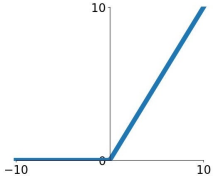
tanh

$$\tanh(x)$$



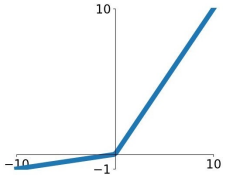
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

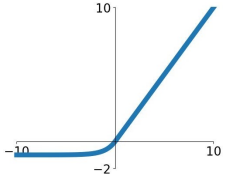


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



[Slide: Andrej Karpathy]

Demo time!

- Link: https://playground.tensorflow.org/

Quiz Time

- Given a neural network and an input, how do you compute its predictions?
  1. You start from the input. Calculate the output of each layer (starting from the first layer), until you get to the output.
  2. You start from the end and make your way to the first layer.

Quiz Time (2)

- What is needed to fully specify a neural network?
  1. Architecture (which input goes through what function etc.)
  2. Parameters of the function (the weights)
  3. Both

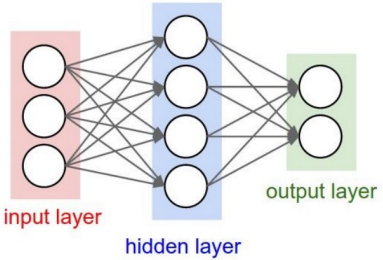
Quiz Time (3)

- What makes neural networks expressive functions?
  1. Activations (non-linearities)
  2. Depth (number of hidden layers)
  3. Width (number of variables in each hidden layer)
  4. All the above

Chapter Plan

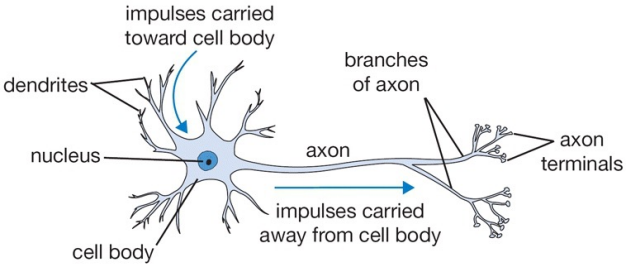
1. Feed-forward networks
2. Neural nets: origins and brief history
3. Word2Vec as a simple neural network
4. Training neural networks: analytical back-propagation
5. Backprop in practice

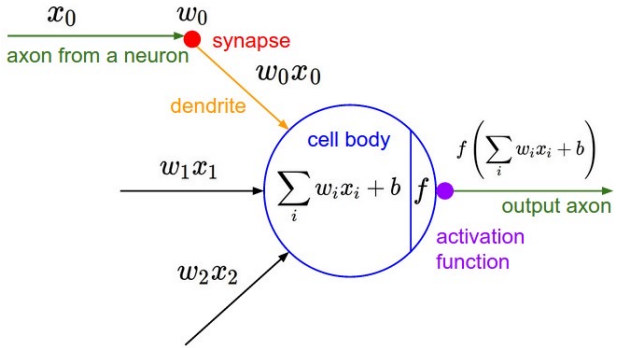
Artificial Neurons: An Inspiration from Nature



- A single node in your neural network
  - Accept information from multiple inputs
  - Transmit information to other neurons

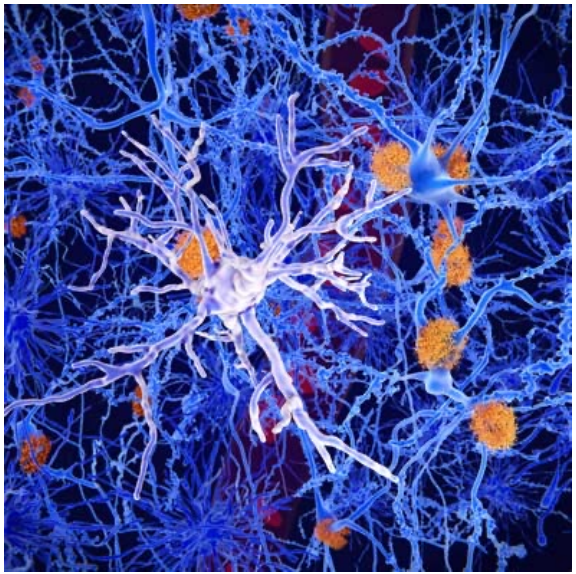
- A neuron's function is inspired by its biological counterpart:
  - Apply some function on inputs signals
  - If output of function over threshold, neuron "fires"



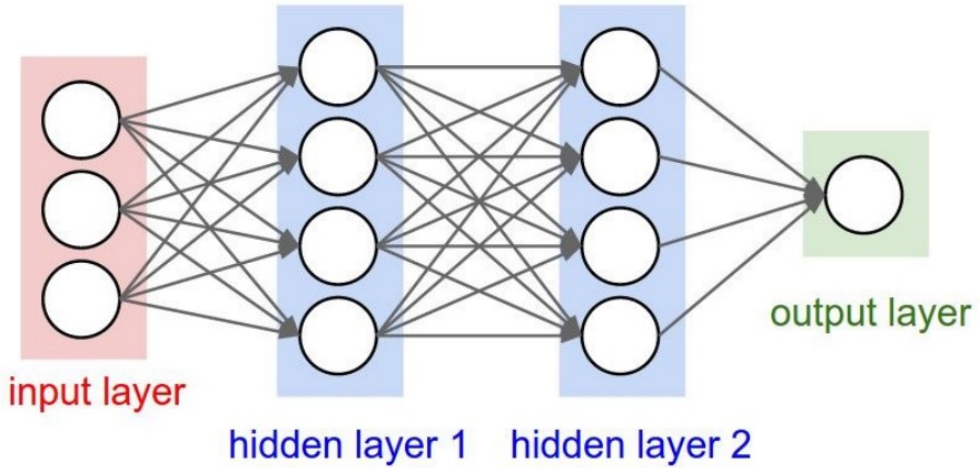


Artificial Neurons: Not Quite Analogous to Nature

Biological neurons:
complex connectivity



Neurons in an artificial neural network:
organized based on a highly regular structure for computational efficiency

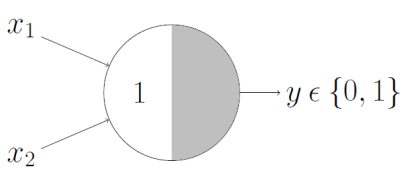


Very Brief History of Neural Networks

1. Single-layer neural networks (1943-1969)
2. Symbolic AI & knowledge engineering (1970-1985)
3. Multi-layer NNs and symbolic learning (1985-1995)
4. chirp chirp …. Statistical learning/probabilistic models (1995-2010)
5. Deep networks and self-supervised learning (2010-?)

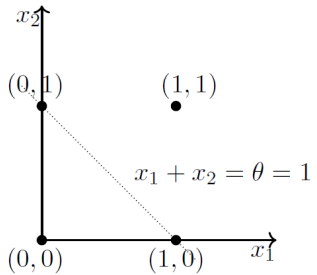
A Neuron as a Mathematical Model of Computation

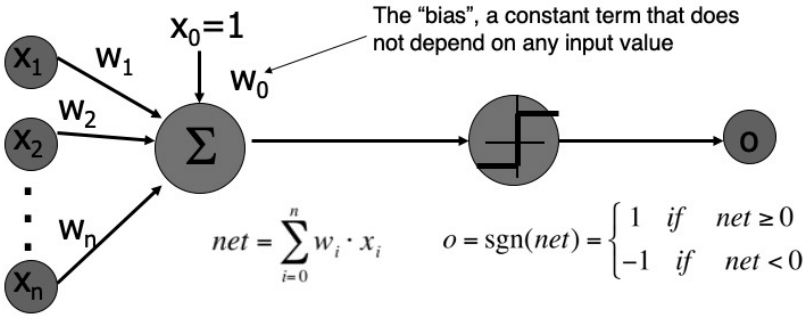
- McCulloch and Pitts (1943) showed how linear threshold units can be used to compute logical functions



OR function

$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 1$$

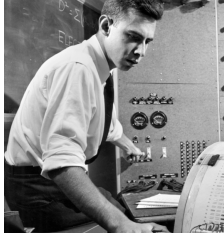




- An alternative model of computation (comparable to "Turing Machine")

[A Logical Calculus of Ideas Immanent in Nervous Activity, McCulloch and Pitts 1943]

Perceptron: Imitating Nature's Learning Process



- Rosenblatt (1959) developed the Perceptron Algorithm, an iterative, hill-climbing algorithm for learning the weights of a linear threshold unit.
  - A single neuron with a fixed input, it can incrementally change weights and learn to produce a fixed output using the Perceptron learning rule.

- Update weights by:

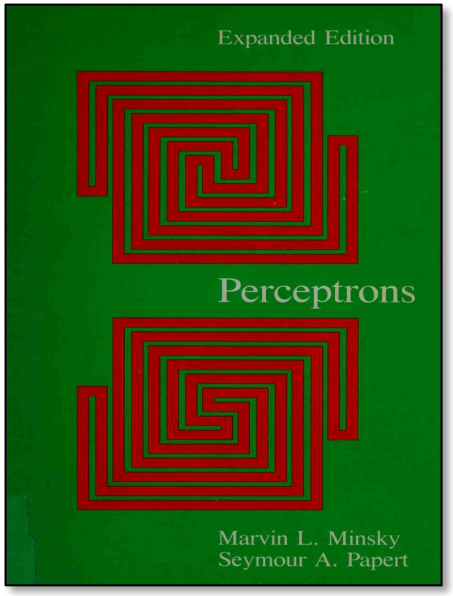
$$w_i = w_i + \eta(t - o)x_i$$

  - where η is the "learning rate," *t* is the teacher output, and *o* is the network output.

  - If output is correct do nothing.
  - If output is higher than *t*, lower weights on active inputs
  - If output is lower than *t*, increase weights on active inputs

[The perceptron: a probabilistic model for information storage and organization in the brain, Rosenblatt 1959]

Perceptron: Demise



- *Perceptons* (1969) by Minsky and Papert illuminated the limitations of the perceptron.

- It showed that:
  - Shallow (2-layer) networks are unable to learn or represent many classification functions (e.g. XOR)
  - Only the linearly separable functions are learnable.
- Also, there was an understanding that deeper networks were infeasible to train.

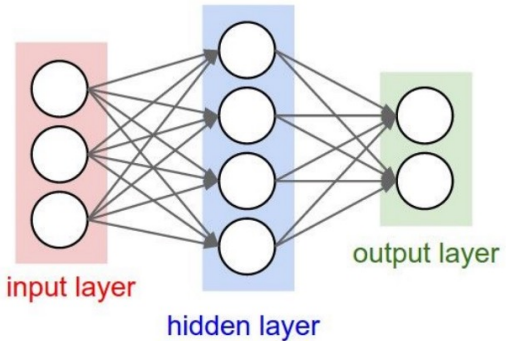
- Result: work on neural-networks dissipated during the 70's and early 80's!

[slide: Ray Mooney]

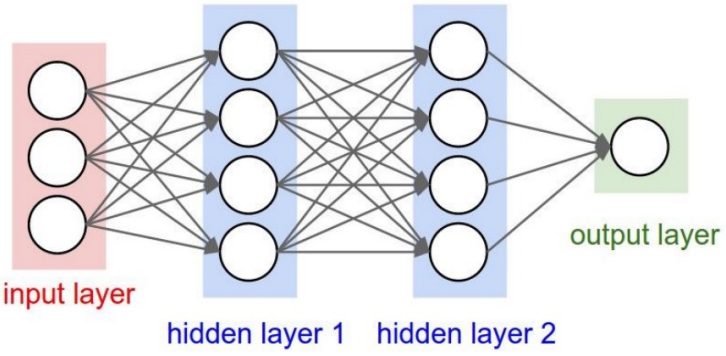
Neural Net Resurgence (1986)

- Interest in NNs revived in the mid 1980's due to the rise of "connectionism."
- Backpropagation algorithm was [re-]introduced for training three-layer NN's.
  - Generalized the iterative "hill climbing" method to approximate networks with multiple layers, but no convergence guarantees.

what was possible
to train in 1970's



what became possible
to train via BP in 1980's



[Learning representations by back-propagating errors, Rumelhart, Hinton & Williams 1986;
for a broader context, see: http://people.idsia.ch/~juergen/who-invented-backpropagation.html]

Second NN Demise (1995-2010)

- Generic Back-Propagation did not generalize that well to training deeper networks.
  - Overfitting / underfitting remained an issue.
  - Computers were still quite slow
- Little theoretical justification for underlying methods.
- Machine learning research moved to graphical/probabilistic models and kernel methods.

[slide: Ray Mooney]

Deep Learning Revolution (2010…)

- Improved methods developed for training deep neural works.

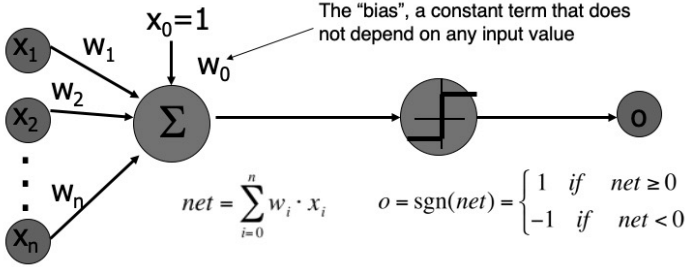
- Particular successes with:
  - Convolutional neural nets (CNNs) for vision (2012 AlexNet showed 16% error reduction on ImageNet).
  - Recurrent neural nets (RNNs) for machine translation and speech recognition.
  - Deep reinforcement learning for game playing (e.g., AlphaGo).

- Massive data and specialized hardware:
  - Large collections of [mostly] supervised (crowdsourced) training data has been critical.
  - Efficient processing of this big data using specialized hardware (Graphics Processing Units, GPUs) has been critical.

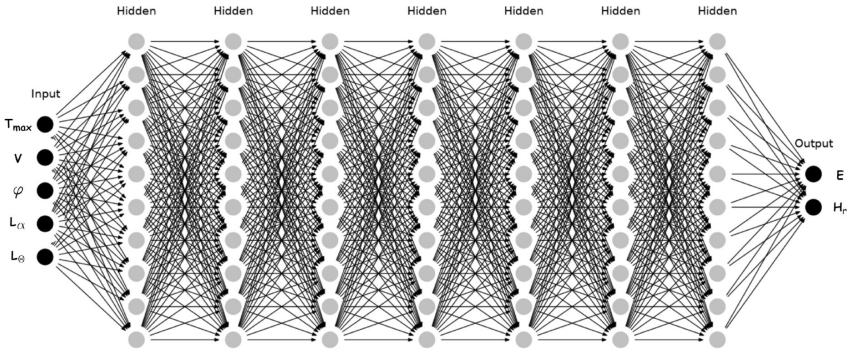
[slide: Ray Mooney]

Deep Learning Revolution (2010…)

How it start



How it's going



[slide: Ray Mooney]

Chapter Plan

1. Feed-forward networks
2. Neural nets: brief history
3. Word2Vec as a simple neural network
4. Training neural networks: analytical back-propagation
5. Backprop in practice

Chapter Plan

1. Feed-forward networks
2. Neural nets: brief history
3. Word2Vec as a simple neural network
4. Training neural networks: analytical back-propagation
5. Backprop in practice

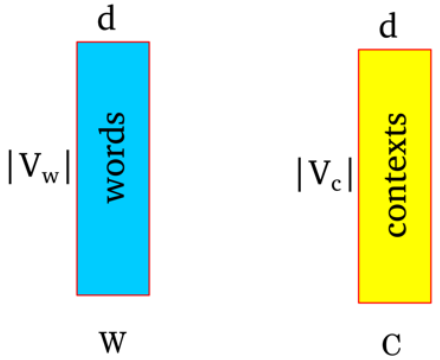
Word2Vec as Neural Net

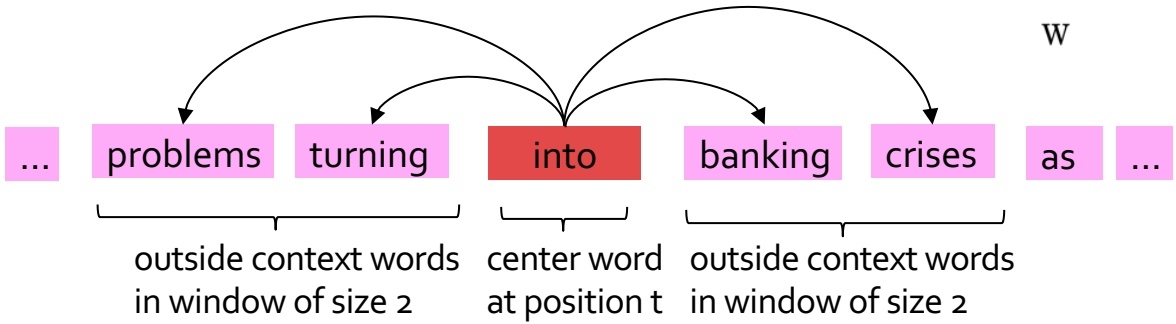
- Word2Vec is actually a very simple neural net!

Word2Vec (SkipGram): Recap

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{x \in V} \exp(u_x^T v_c)}$$

- **Objective:** find embeddings such that the words that co-occur are close.
- Represent each word as a ***d*** dimensional vector.
- Represent each context as a ***d*** dimensional vector.

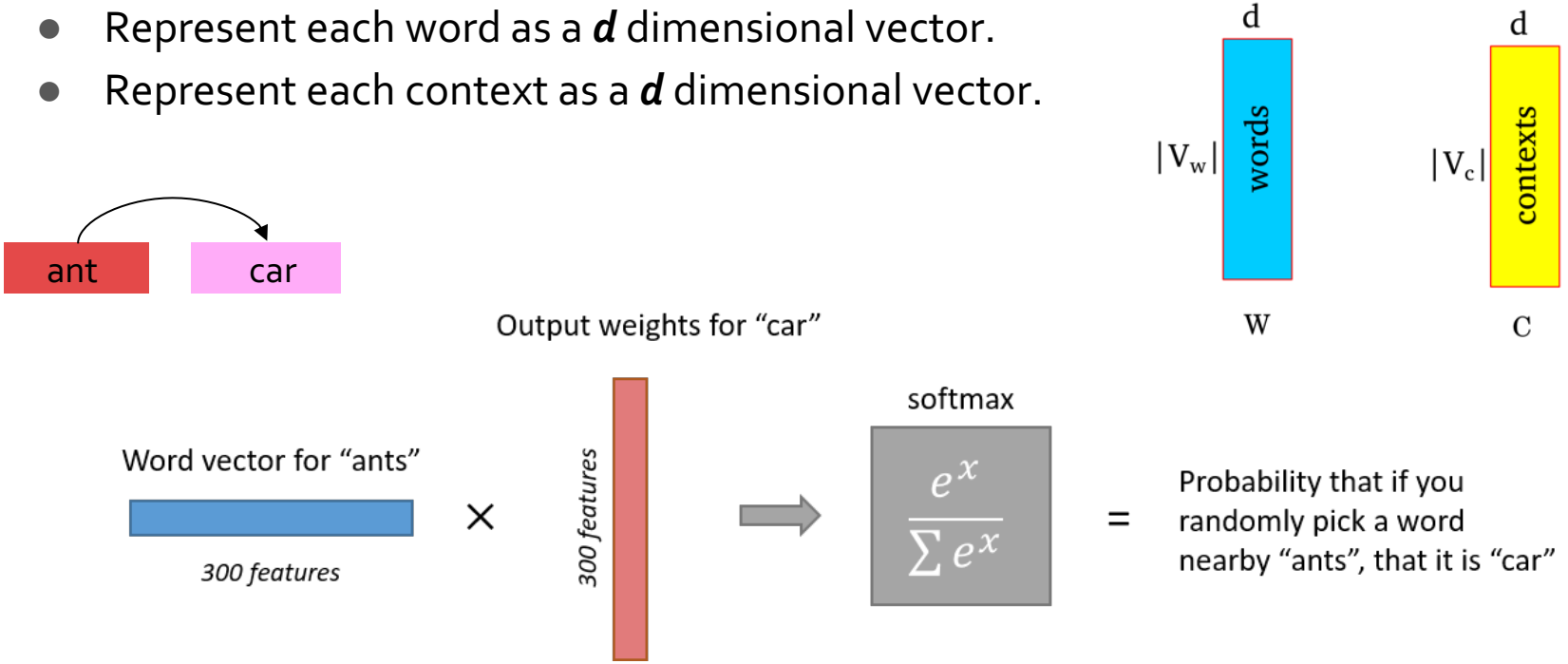




Word2Vec (SkipGram): Recap

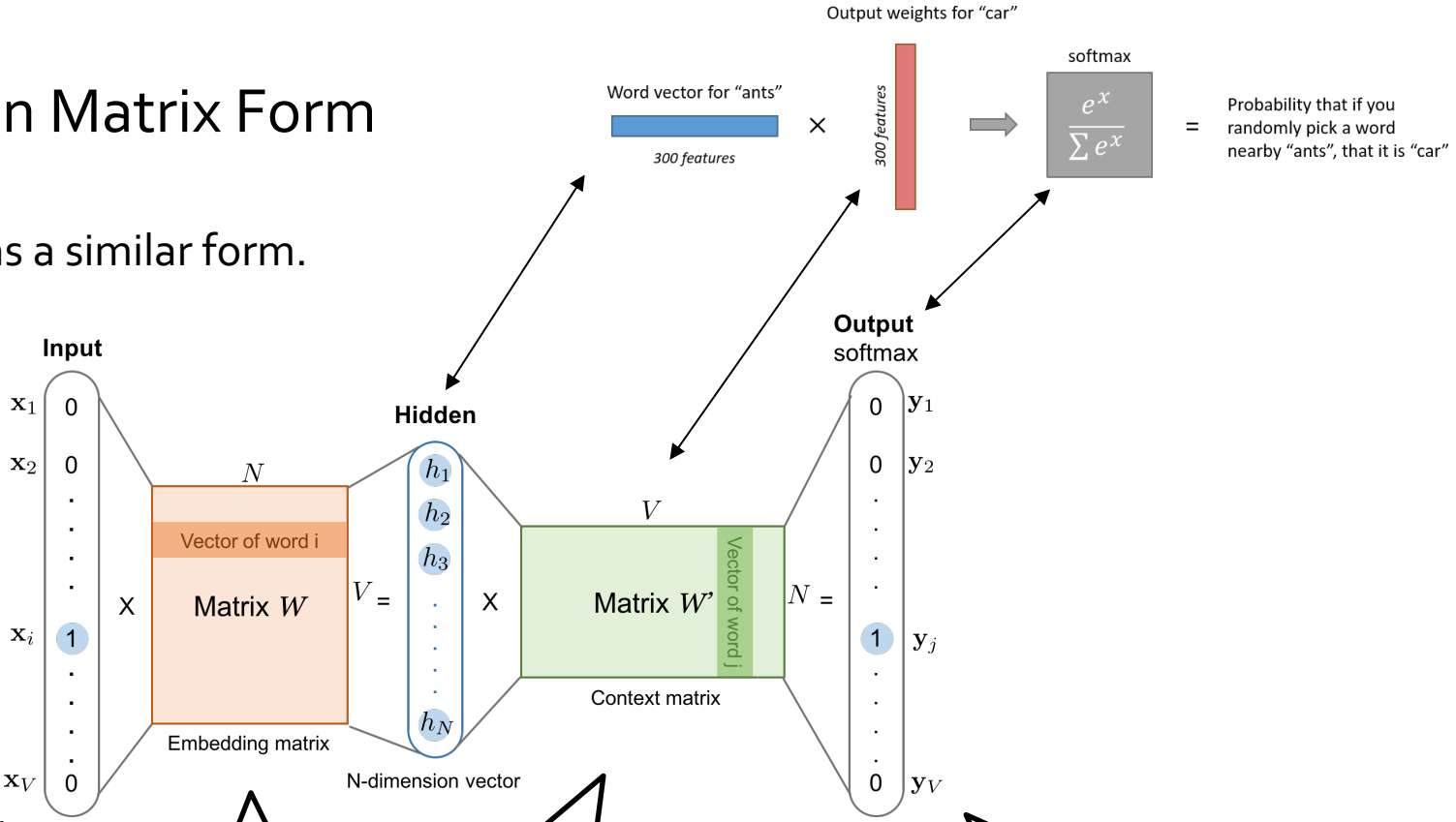
$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{x \in V} \exp(u_x^T v_c)}$$

- **Objective:** find embeddings such that the words that co-occur are close.
- Represent each word as a ***d*** dimensional vector.
- Represent each context as a ***d*** dimensional vector.



SkipGram in Matrix Form

CBOW also has a similar form.



A one-hot vector for "ants"

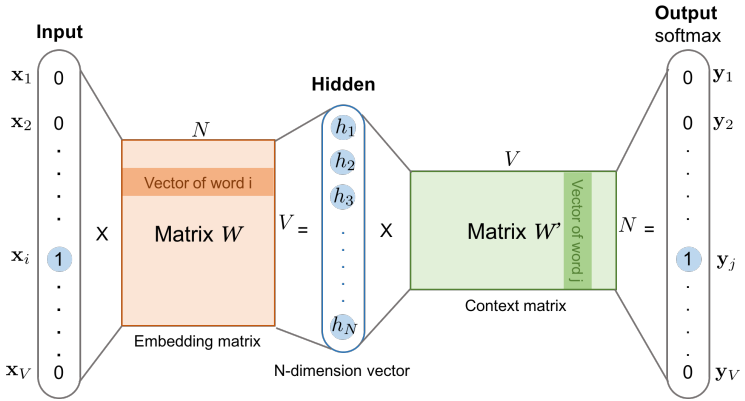
Embedding for "ants"

Dot product of the "ant" emb w/ emb of other words

Prob assigned to each word appearing in the context

Word2Vec as Neural Network

- Word2Vec is basically a very simple feed-forward neural network!



- Any algorithms for neural networks could also be applied to Word2Vec.
  - For example, we can use Backpropagation (forthcoming!) to train Word2Vec. (HW3!!)
- We can extend W2V by modifying the architecture.
  - Contextual self-supervised representations: next chapter!

[figure: https://lilianweng.github.io/posts/2017-10-15-word-embedding/]

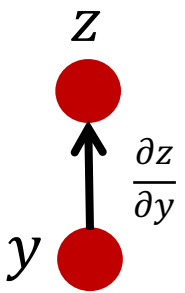
Chapter Plan

1. Feed-forward networks
2. Neural nets: brief history
3. Word2Vec as a simple neural network
4. Training neural networks: back-propagation
5. Backprop in practice

Algebra Refresher 

Derivatives

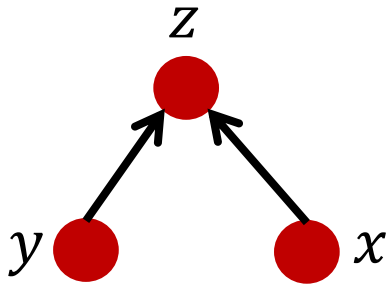
- First let's get the notation right:
- The arrow shows functional dependence of z on y, i.e. given y, we can calculate z.
  - For example: $z(y) = 2y^2$
- The derivative of z, with respect to y: $\frac{\partial z}{\partial y}$



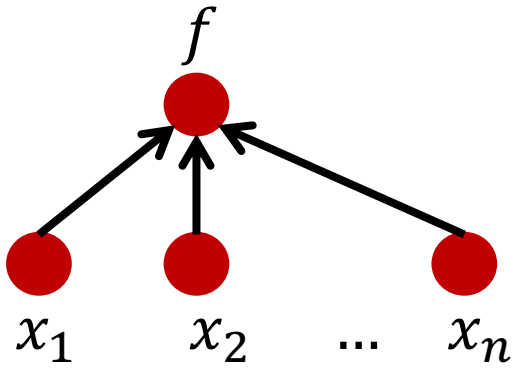
Quiz time!

- If $z(x, y) = y^4 x^5$ what is the following derivative $\frac{\partial z}{\partial y}$?

1. $\frac{\partial z}{\partial y} = 4y^3 x^5$
2. $\frac{\partial z}{\partial y} = 5y^4 x^4$
3. $\frac{\partial z}{\partial y} = 20y^3 x^4$
4. None of the above



Gradient



- Given a function with 1 output and n inputs

$$f(\mathbf{x}) = f(x_1, x_2, \ldots, x_n) \in \mathbb{R}$$

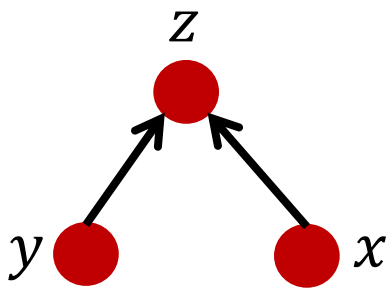
- Its gradient is a vector of partial derivatives with respect to each input

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \in \mathbb{R}^n$$

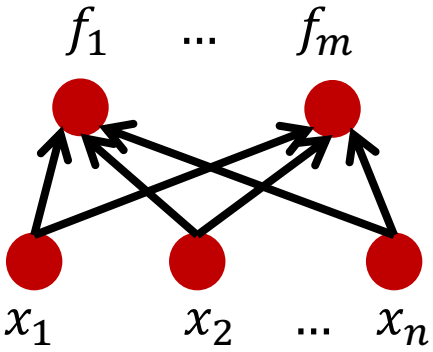
(always assume vectors are column vectors, i.e., they're in $\mathbb{R}^{n \times 1}$)

Quiz time!

- If $z(x, y) = y^4 x^5$ what is the following gradient ∇z?
  1. $\nabla z(x, y) = 4y^3 x^5$
  2. $\nabla z(x, y) = (5y^4 x^4, 20y^3 x^4)$
  3. $\nabla z(x, y) = (5y^4 x^4, 4y^3 x^5)$
  4. None of the above



Jacobian Matrix: Generalization of the Gradient



- Given a function with ***m* outputs** and ***n* inputs**

$$\mathbf{f}(\mathbf{x}) = [f_1(x_1, x_2, \dots, x_n), \dots, f_m(x_1, x_2, \dots, x_n)] \in \mathbb{R}^m$$

- It's Jacobian is an ***m* x *n* matrix** of partial derivatives: $\left(\mathbf{J_f}(\mathbf{x})\right)_{ij} = \frac{\partial f_i}{\partial x_j}$

$$\mathbf{J_f}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \dots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{m \times n} \quad \text{or} \quad \left(\mathbf{J_f}(\mathbf{x})\right)_{ij} = \frac{\partial f_i}{\partial x_j}$$

- When m=1 (scalar-valued function), Jacobian reduces to $\nabla^{\mathrm{T}}\mathbf{f}(\mathbf{x})$ (gradient transpose).
- When m=n=1 (single-variable function), Jacobian reduces to the derivative of $\mathbf{f}$.

Jacobian for Matrix Inputs

- Given a function with $\boldsymbol{m}$ **outputs** and $\boldsymbol{n}\times\boldsymbol{p}$ **inputs**

$$\mathbf{f}(\mathbf{X}) = [f_1(\mathbf{X}), \ldots, f_m(\mathbf{X})] \in \mathbb{R}^m, \text{ where } \mathbf{X} = \begin{bmatrix} x_{11} & \cdots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{bmatrix} \in \mathbb{R}^{n\times p}$$

- Jacobian is a $\mathrm{m}\times n\times p$ **tensor** (i.e., matrix of matrices) of partial derivatives:

$$\left(\mathbf{J}_\mathbf{f}(\mathbf{X})\right)_{ijk} = \frac{\partial f_i}{\partial x_{jk}}$$

- The Jacobian math holds if you keep adding more dimensions to the input or output.

Why Use Matrix/Tensor Form?

In essence, matrix form (multi-variate calculus) is just an extension of single-variable calculus.

Two reasons:

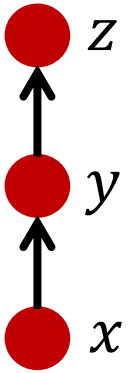
- Compact derivations: with matrix form calculations we can compute a concise statements.
- Implementing algorithms in matrix form is much faster.
  - GPUs are optimized for VERY FAST matrix/tensor operations.

Chain Rule

- Function composition:

$$z \circ y(x) = z\big(y(x)\big) = z(x)$$

If z is a function of y, and y is a function of x, then z is a function of x, as well.



Then:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

Chain Rule for Multivariable Functions

- Let $\mathbf{x} \in \mathbb{R}^d$, $\mathbf{g}$: $\mathbb{R}^d \to \mathbb{R}^n$, $\mathbf{f}$: $\mathbb{R}^n \to \mathbb{R}^m$
- Composing them: $\mathbf{f} \circ \mathbf{g}(\mathbf{x}) = \mathbf{f}\big(\mathbf{g}(\mathbf{x})\big): \mathbb{R}^d \to \mathbb{R}^m$

The result looks similar to the single-variable setup:

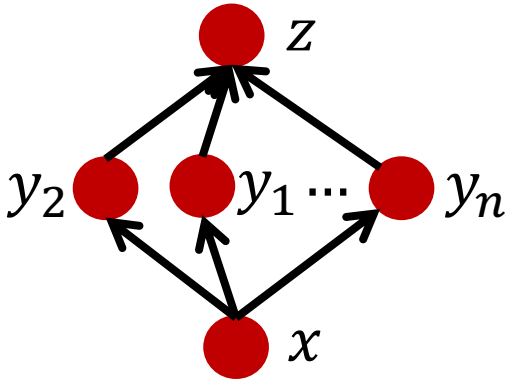
$$\mathbf{J}_{\mathbf{f}\circ\mathbf{g}}(\mathbf{x}) = \mathbf{J}_{\mathbf{f}}(\mathbf{g}(\mathbf{x}))\ \ \mathbf{J}_{\mathbf{g}}(\mathbf{x})$$

Note, the above statement is a **matrix** multiplication!

Function $\mathbf{f} \circ \mathbf{g}$ has *m* outputs and *d* inputs → *m* by *d* Jacobian

Quiz Time!

Let $x \in \mathbb{R}$, $\mathbf{y}: \mathbb{R} \to \mathbb{R}^n$, $\mathbf{z}: \mathbb{R}^n \to \mathbb{R}$



What is the Jacobean of $z \circ \mathbf{y}(x) = z(y_1(x), \ldots, y_n(x))$?

1. $\mathbf{J}_{z\circ\mathbf{y}}(x) = \mathbf{J}_z(\mathbf{y}(x))\ \mathbf{J}_\mathbf{y}(x)$
2. $\mathbf{J}_{z\circ\mathbf{y}}(x) = \left[\frac{\partial z}{\partial y_1}, \ldots, \frac{\partial z}{\partial y_n}\right]\left[\frac{\partial y_1}{\partial x}, \ldots, \frac{\partial y_n}{\partial x}\right]^{\mathrm{T}}$
3. $\mathbf{J}_{z\circ\mathbf{y}}(x) = \sum_{\mathrm{i}=1}^{n} \frac{\partial z}{\partial y_i}\ \frac{\partial y_i}{\partial x}$
4. All the above!

We will stop here!

- Today: lots of background about neural networks!

- Next time: training a neural net!