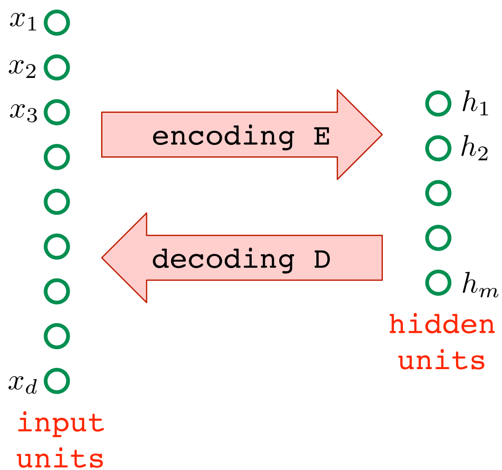
Autoencoders

Topics we'll cover

1. Autoencoders
2. k-means and PCA as autoencoders
3. Manifold learning
4. Independent component analysis
5. Stacked autoencoders

Autoencoders

Finding the **underlying degrees of freedom** of data

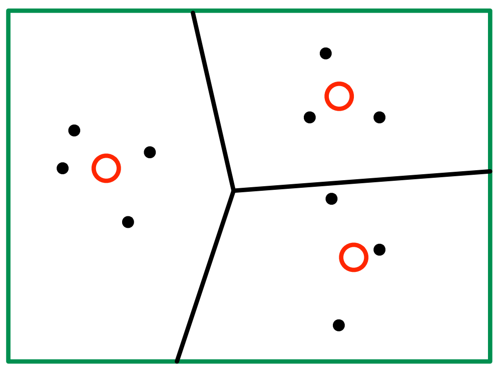


Ideally $x \approx D(E(x))$ on data points $x \in \mathbb{R}^d$

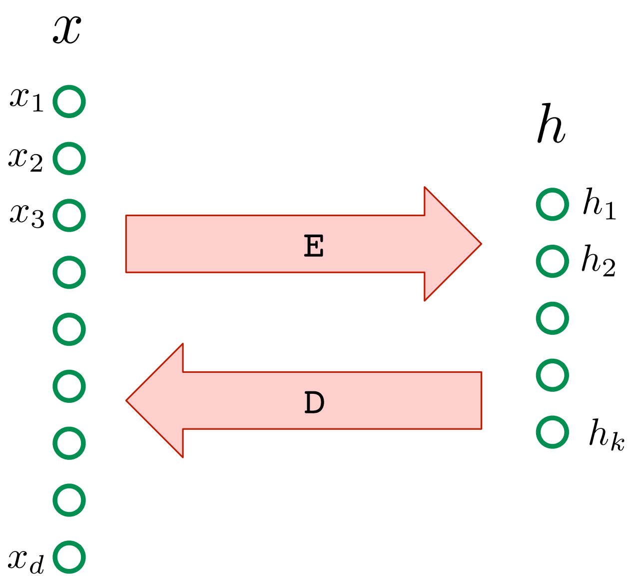
The k-means clustering scheme, revisited

The k-means problem:

- Given: $x^{(1)}, \ldots, x^{(n)} \in \mathbb{R}^d$; integer k
- Find: k centers $\mu_1, \ldots, \mu_k \in \mathbb{R}^d$ that minimize $\sum_{i=1}^n \min_{1 \leq j \leq k} \|x^{(i)} - \mu_j\|^2$



The k-means autoencoder



Principal component analysis, revisited

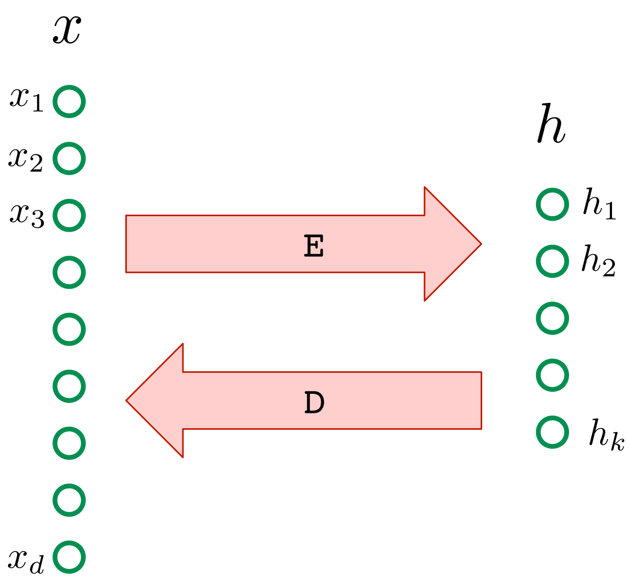
The PCA problem:

- Given: $x^{(1)}, \ldots, x^{(n)} \in \mathbb{R}^d$; integer k
- Find: the projection $\mathbb{R}^d \to \mathbb{R}^k$ that maximizes the variance of the projected data

Solution:

- Compute the covariance matrix of the data
- Let $u_1, \ldots, u_k$ be the top k eigenvectors of this matrix
- Let $k \times d$ matrix U have the u_i as its columns
- Projection: $x \mapsto U^T x$
- Reconstruction: $z \mapsto Uz$

The PCA autoencoder



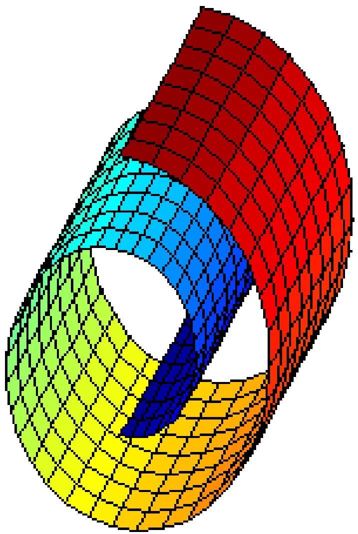
Some other types of intrinsic structure

1. Manifold learning
   The data lies on a k-dimensional manifold.

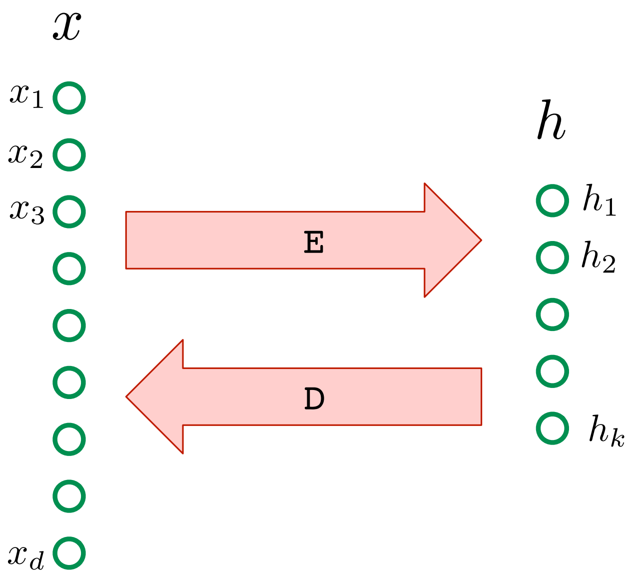
2. Independent component analysis
   The data are linear combinations of hidden features that are independent.

Manifold learning

Sometimes data in a high-dimensional space $\mathbb{R}^d$ in fact lies close to a k-dimensional manifold, for $k \ll d$

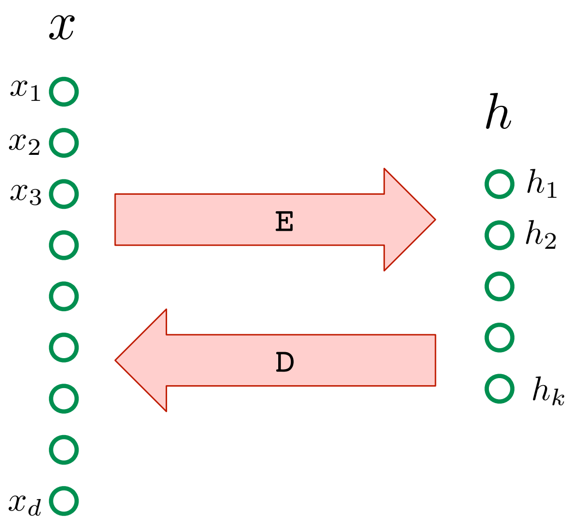


The manifold autoencoder

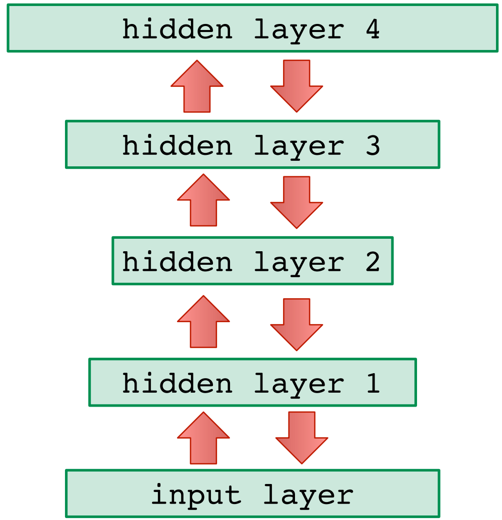


Independent component analysis

The cocktail party problem



Stacked autoencoders



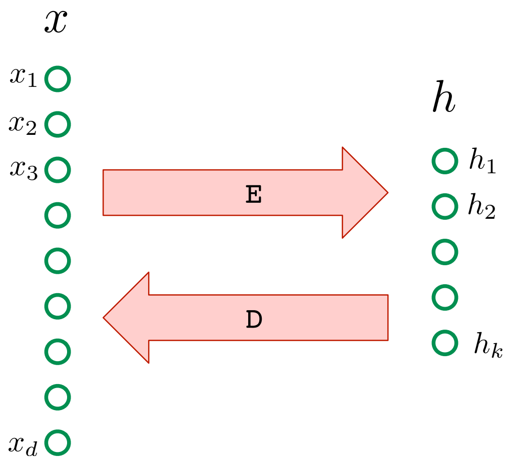
- Fit one layer at a time to the previous layer's activations
- Then fine-tune whole structure to minimize reconstruction error

Distributed representations

Topics we'll cover

1. One-hot versus distributed encodings
2. Word embeddings

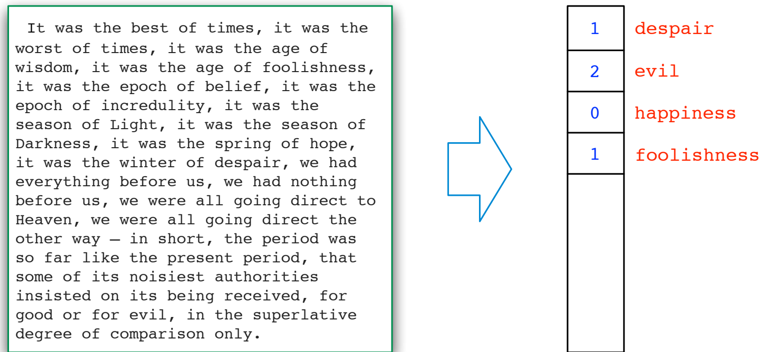
One-hot versus distributed representations



- *k*-means: **one-hot** encoding
- PCA: **distributed** encoding

The bag-of-words representation

One-hot encoding of words:



- Fix V = some vocabulary.
- Treat each sentence (or document) as a vector of length $|V|$:

$$x = (x_1, x_2, \ldots, x_{|V|}),$$

where $x_i = \#$ of times the ith word appears in the sentence.

Word co-occurrences

You shall know a word by the company it keeps. (J.R. Firth, 1957)

- Much of the meaning of a word w is captured by the words it co-occurs with:

$$w_1 \quad w_2 \quad w_3 \quad w \quad w_4 \quad w_5 \quad w_6$$

- Find an embedding of words based on these co-occurrences.

A simple approach to word embedding

Fix a vocabulary V. Then, using a corpus of text:

1. Look at each word w and its surrounding *context*: $w_1 \quad w_2 \quad w_3 \quad w \quad w_4 \quad w_5 \quad w_6$
   - $n(w, c) = \#$ times word c occurs in the context of word w
   - Yields a probability distribution $\Pr(c|w)$.

2. Positive pointwise mutual information:

$$\Phi_c(w) = \max\left(0, \log \frac{\Pr(c|w)}{\Pr(c)}\right)$$

   This is a $|V|$-dimensional representation of word w.

3. Reduce dimension using PCA.

The embedding

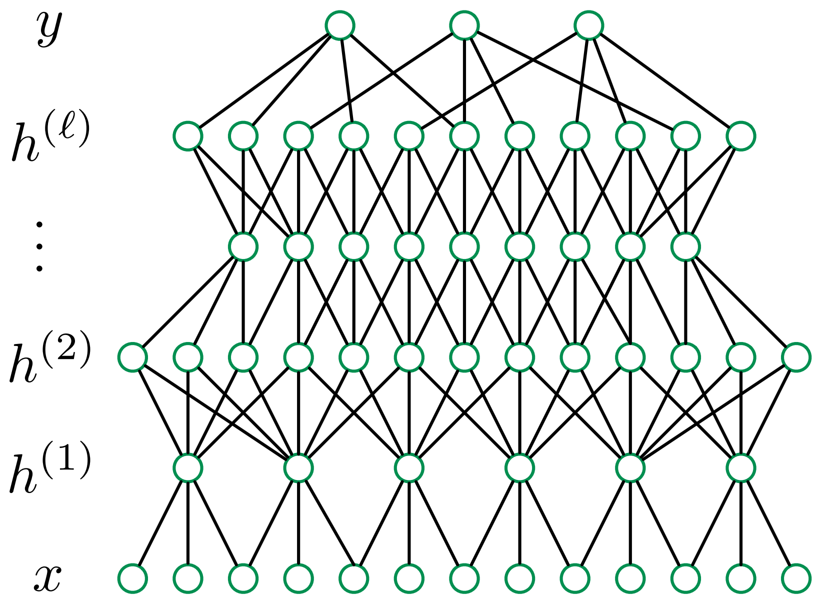
- Which word's vector is closest to that of `Africa`?
  `Asia`
- Solving analogy problems: `king` is to `queen` as `man` is to ?
  - $\text{vec}(\texttt{king}) - \text{vec}(\texttt{queen}) = \text{vec}(\texttt{man}) - \text{vec}(?)$
  - $\text{vec}(?) = \text{vec}(\texttt{man}) - \text{vec}(\texttt{king}) + \text{vec}(\texttt{queen})$
  - Nearest neighbor of this vector is $\text{vec}(\texttt{woman})$.

Feedforward neural nets

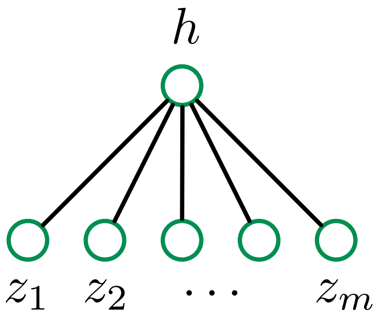
Topics we'll cover

1. The architecture
2. The functions
3. The effect of depth

The architecture



The value at a hidden unit

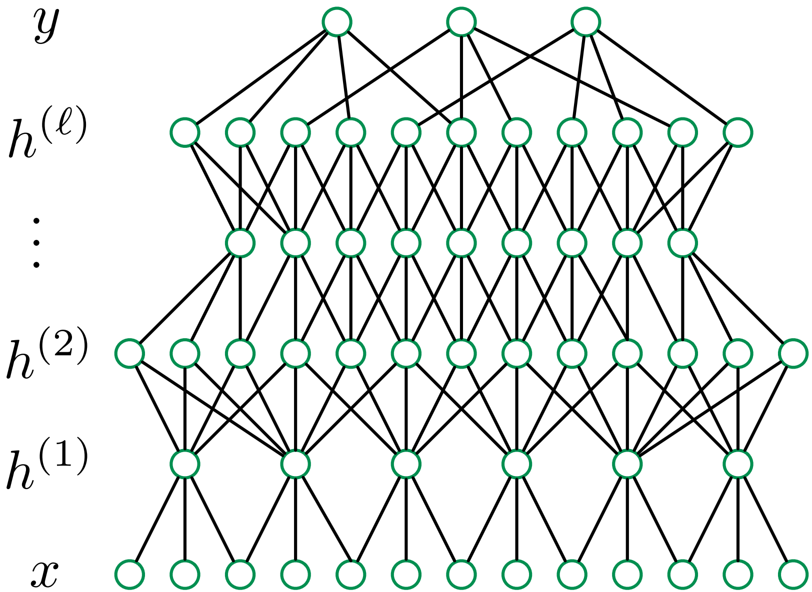


How is h computed from $z_1, \ldots, z_m$?

- $h = \sigma(w_1 z_1 + w_2 z_2 + \cdots + w_m z_m + b)$
- $\sigma(\cdot)$ is a nonlinear **activation function**, e.g. "rectified linear"

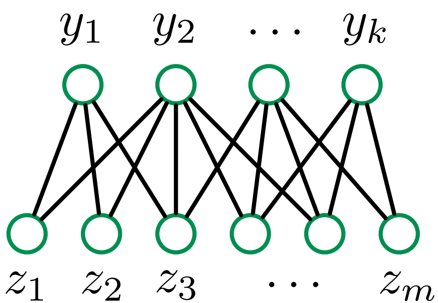
$$\sigma(u) = \begin{cases} u & \text{if } u \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Why do we need nonlinear activation functions?



The output layer

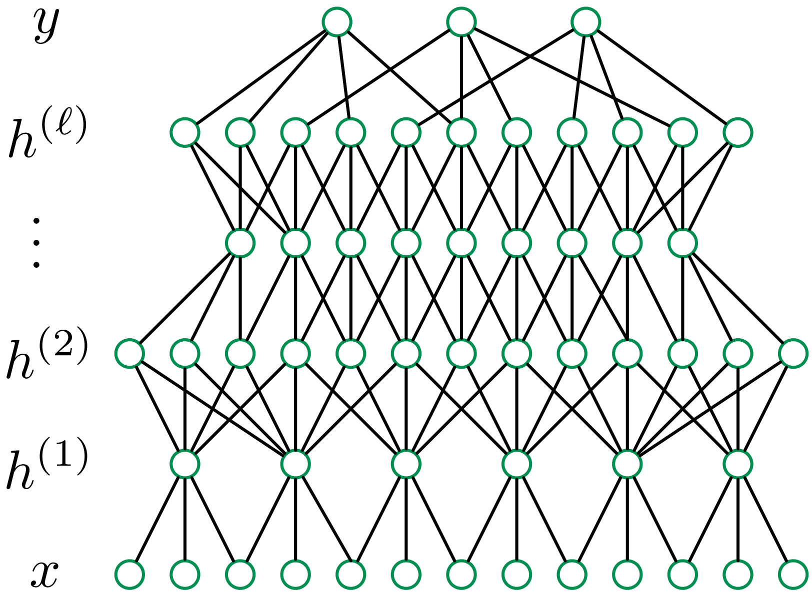
Classification task with k labels: want k probabilities summing to 1.



- $y_1, \ldots, y_k$ are linear functions of the parent nodes z_i.
- Get probabilities using **softmax**:

$$\Pr(\text{label } j) = \frac{e^{y_j}}{e^{y_1} + \cdots + e^{y_k}}.$$

The complexity



The effect of depth

- Universal approximator
  Any function can be arbitrarily well approximated by a neural net with one hidden layer.

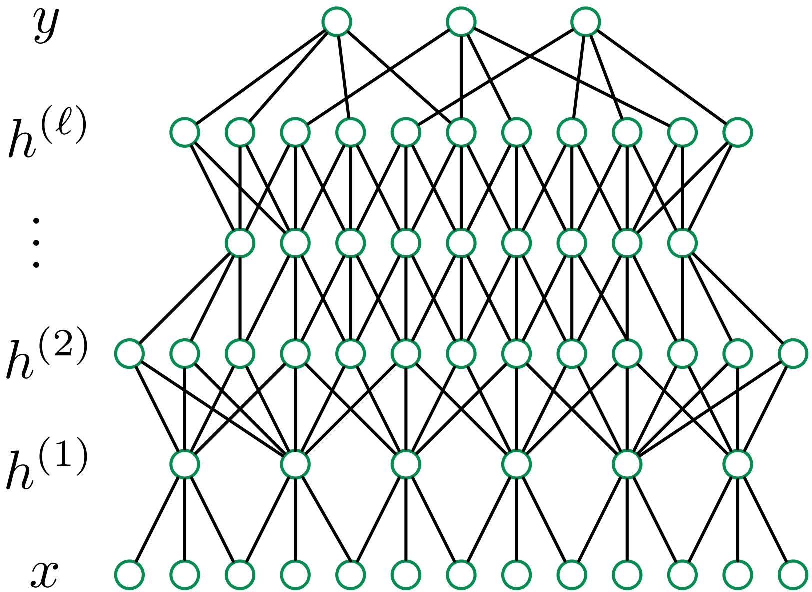
- Concerns about size
  To fit certain classes of functions:
  - Either: one hidden layer of enormous size
  - Or: multiple hidden layers of moderate size

Training a feedforward neural net

Topics we'll cover

1. The loss function
2. Back-propagation
3. Early stopping and dropout

Feedforward nets



The loss function

Classification problem with k labels.

- Parameters of entire net: W
- For any input x, net computes probabilities of labels:

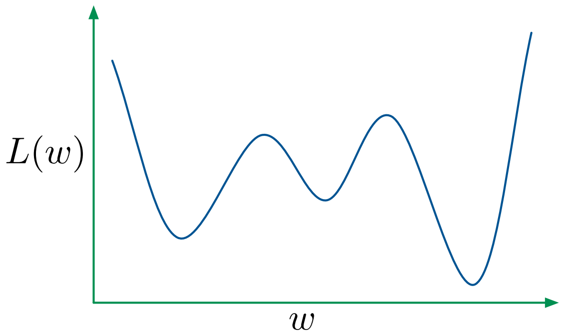
$$\Pr_W(\text{label} = j | x)$$

- Given data set $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$, loss function:

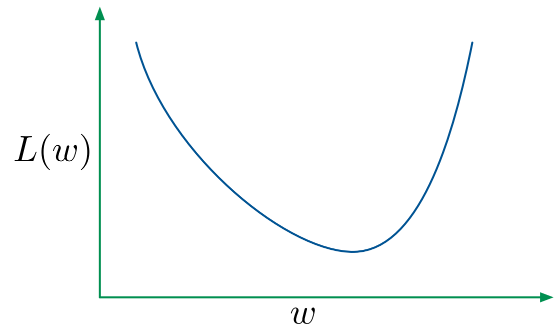
$$L(W) = -\sum_{i=1}^{n} \ln \Pr_W(y^{(i)} | x^{(i)})$$

(sometimes called **cross-entropy**).

Nature of the loss function



versus



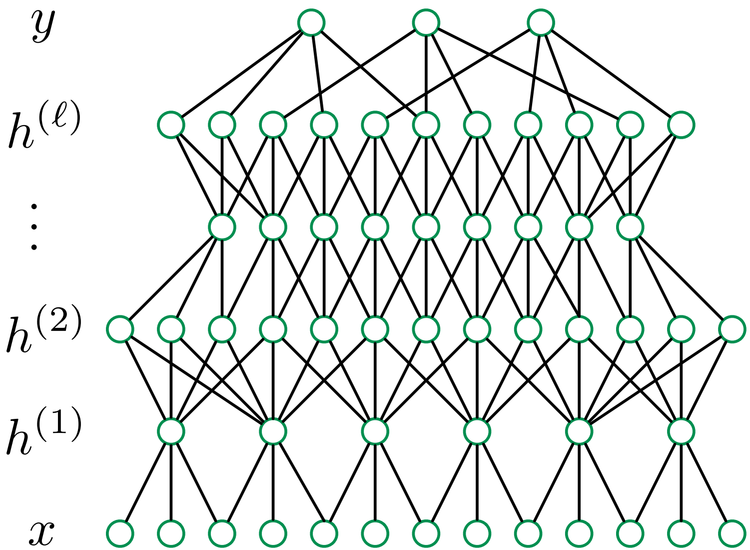
Variants of gradient descent

Initialize W and then repeatedly update.

1. Gradient descent
   Each update involves the entire training set.
2. Stochastic gradient descent
   Each update involves a single data point.
3. Mini-batch stochastic gradient descent
   Each update involves a modest, fixed number of data points.

Derivative of the loss function

Update for a specific parameter: derivative of loss function wrt that parameter.



Chain rule

1. Suppose $h(x) = g(f(x))$, where $x \in \mathbb{R}$ and $f, g : \mathbb{R} \to \mathbb{R}$.

   Then: $h'(x) = g'(f(x))\, f'(x)$

2. Suppose z is a function of y, which is a function of x.

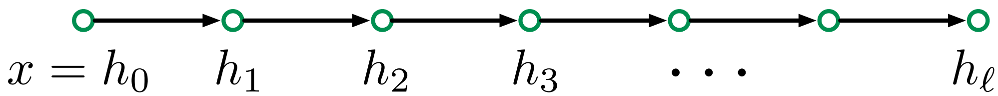
$$x \longrightarrow y \longrightarrow z$$

   Then:

$$\frac{dz}{dx} = \frac{dz}{dy}\,\frac{dy}{dx}$$

A single chain of nodes

A neural net with one node per hidden layer:



For a specific input x,

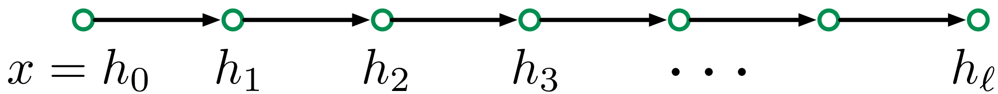
- $h_i = \sigma(w_i h_{i-1} + b_i)$
- The loss L can be gleaned from h_ℓ

To compute dL/dw_i we just need dL/dh_i:

$$\frac{dL}{dw_i} = \frac{dL}{dh_i}\,\frac{dh_i}{dw_i} = \frac{dL}{dh_i}\,\sigma'(w_i h_{i-1} + b_i)\,h_{i-1}$$

Backpropagation

- On a single forward pass, compute all the h_i.
- On a single backward pass, compute $dL/dh_\ell, \ldots, dL/dh_1$



From $h_{i+1} = \sigma(w_{i+1} h_i + b_{i+1})$, we have

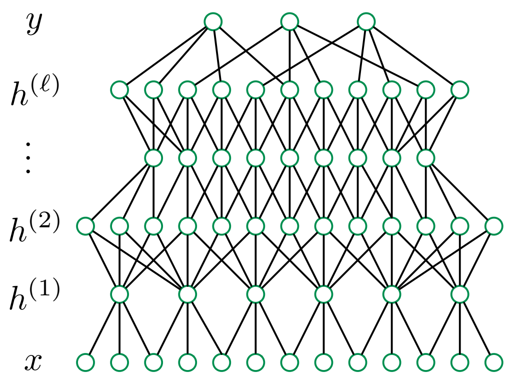
$$\frac{dL}{dh_i} = \frac{dL}{dh_{i+1}} \frac{dh_{i+1}}{dh_i} = \frac{dL}{dh_{i+1}} \sigma'(w_{i+1} h_i + b_{i+1})\, w_{i+1}$$

Improving generalization

1. **Early stopping**
   - Validation set to better track error rate
   - Revert to earlier model when recent training hasn't improved error

2. **Dropout**

   During training, delete each hidden unit with probability 1/2, independently.



What we skipped

Probabilistic approaches to machine learning

1. Graphical models
2. Causality
3. Bayesian methods

Reinforcement learning

The human side of machine learning

1. Trust
2. Transparency
3. Explanations