Statistical Language Models

Week 8

NASA

- NASA LDA with 24 topics
- library(topicmodels)

- # be aware that running this model is time intensive
- desc_lda = LDA(desc_dtm, k = 24, control = list(seed = 1234))
- desc_lda

- tidy_lda <- tidy(desc_lda)

- tidy_lda

Their β = probability of a term given a topic

- top_terms <- tidy_lda %>%
- group_by(topic) %>%
- top_n(10, beta) %>%
- ungroup() %>%
- arrange(topic, -beta)
- top_terms

In plots

- top_terms %>%
- mutate(term = reorder_within(term, beta, topic)) %>%
- group_by(topic, term) %>%
- arrange(desc(beta)) %>%
- ungroup() %>%
-
-
- ggplot(aes(term, beta, fill = as.factor(topic))) +
- geom_col(show.legend = FALSE) +
- coord_flip() +
- scale_x_reordered() +
- labs(title = "Top 10 terms in each NASA topic",
- x = NULL, y = expression(beta)) +
- facet_wrap(~ topic, ncol = 4, scales = "free")

Their γ = probability of topic within document Connecting topic modeling with keywords

- lda_gamma = tidy(desc_lda, matrix = "gamma")

- lda_gamma

- lda_gamma = full_join(lda_gamma, nasa_keyword, by = c("document" = "id"))

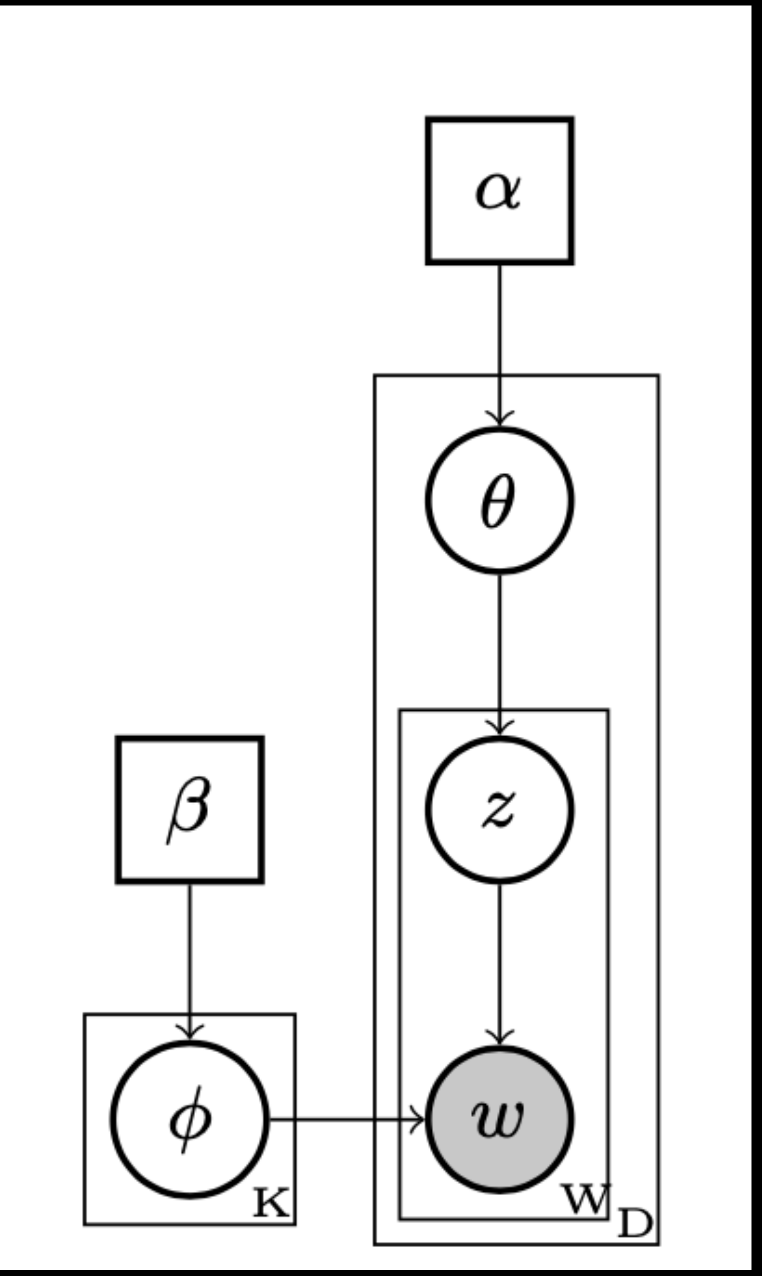
- lda_gamma

In plots

- top_keywords <- lda_gamma %>%
-   filter(gamma > 0.9) %>%
-   count(topic, keyword, sort = TRUE)
- top_keywords

- top_keywords %>%
-   group_by(topic) %>%
-   top_n(5, n) %>%
-   ungroup %>%
-   mutate(keyword = reorder_within(keyword, n, topic)) %>%
-   ggplot(aes(keyword, n, fill = as.factor(topic))) +
-   geom_col(show.legend = FALSE) +
-   labs(title = "Top keywords for each topic",
-        x = NULL, y = "Number of documents") +
-   coord_flip() +
-   scale_x_reordered() +
-   facet_wrap(~ topic, ncol = 4, scales = "free")

Observed and Latent



- Observed: Words in Documents
- Latent: topics for words and documents
- Exploration of Latent space gives interesting insights

Word2Vec

- Neural Network model to predict a word from those around it
- Produces a numeric latent (embedding) vector space
- https://arxiv.org/pdf/1301.3781.pdf

Matrix Version (not often used in practice)
Based on https://iksinc.online/tag/continuous-bag-of-words-cbow/

- Training corpus: {“the dog saw a cat”,
- “the dog chased the cat”,
- “the cat climbed a tree”}
- Size of vocabulary? (This defined input and output dimensions)
- Define a latent (embedding) of dimension 3 for our vector space

Model fitting

- Initialize input weight matrix and output weight matrix such that for vocabulary dimension V and hidden dimension H we have matrices:
- $\mathrm{INPUT}_{\{VxH\}}$
- $\mathrm{OUTPUT}_{\{HxV\}}$
- Ordering the words alphabetically, the input vector representing "dog":
- X = [0,…,0,1,0…0]

Model fitting

- Initialize input weight matrix and output weight matrix such that for vocabulary dimension V and hidden dimension H we have matrices:
- $INPUT_{\{VxH\}}$
- $OUTPUT_{\{HxV\}}$
- Ordering the words alphabetically, the input vector representing "dog":
- X = [0,…,0,1,0…0]

- Vocab = c("a","cat","chased","climbed","dog","saw","the","tree")
- length(Vocab)
- V = length(Vocab)
- H = 3

- INPUT = matrix(rnorm(V*H),nrow=V,ncol = H)
- OUTPUT = matrix(rnorm(V*H),nrow=H,ncol = V)
- #dog:
- X = rep(0,V)
- X[Vocab=="dog"]=1

- #Hiden Layer (transposed)
- tH = t(X)%*%INPUT

- #Output from the model (no weights trained)
- Out = tH%*% OUTPUT
- rbind(Vocab,Out)

- #Convert to probabilities
- P(word = w | word context) = $\frac{exp[activation(w)]}{\sum_{v \in Vocab} exp[activation(v)]}$
- Probs = exp(Out)/sum(exp(Out))
- rbind(Vocab,Out,Probs)

Model could be optimized

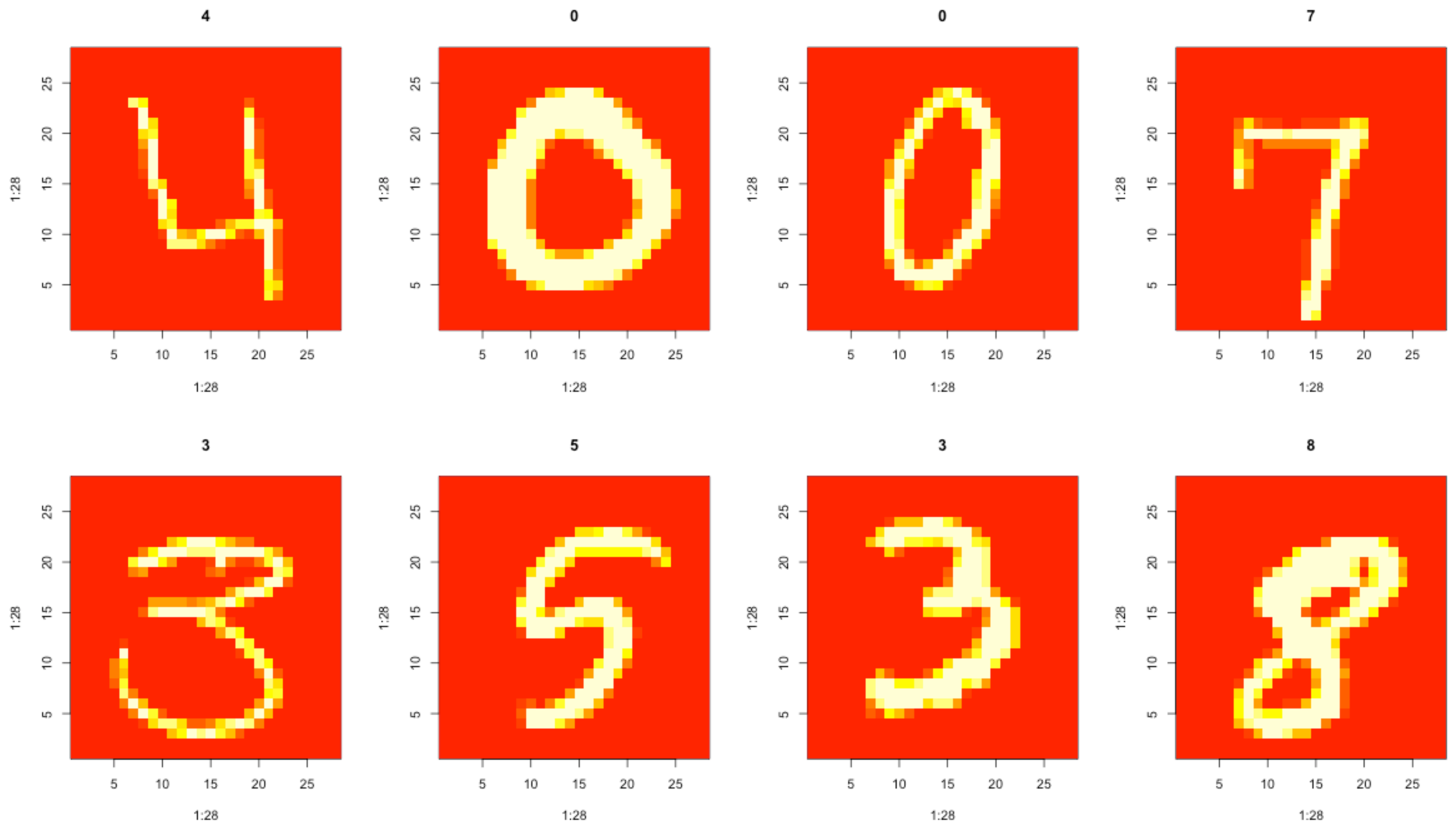
- Hidden layer is a lower dimension vector space.

Neural Networks

- Massively hierarchical non-parametric models thought to work by magic

Mixed National Institute of Standards and Technology database (MNIST) database

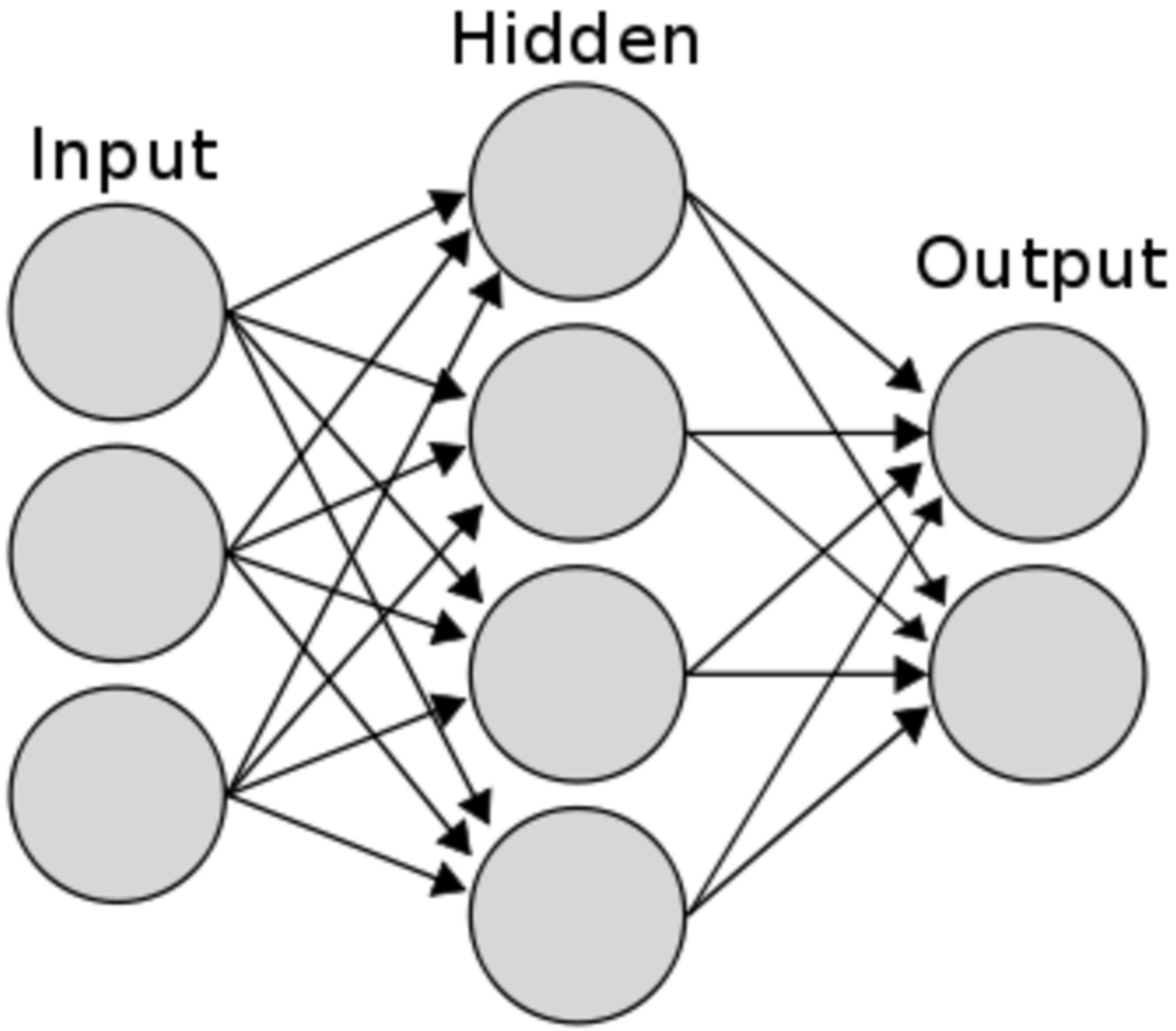
- 28 x 28 pixel images = 784 variables
- 10 different digits to be classified
- A **mix** of American Census Bureau employees and American high school students
- Goal is to classify these digits to human accuracy but faster and cheaper than humans can do it.



- Identify the wine producer from different wine traits.
- 12 wine variables:
- 1) Alcohol 2) Malic acid 3) Ash
  4) Alcalinity of ash 5) Magnesium
  6) Total phenols 7) Flavanoids
  8) Nonflavanoid phenols 9) Proanthocyanins
  10)Color intensity 11)Hue 12)OD280/OD315 of diluted wines 13)Proline

Species identification

- Iris data set is the kindergarten of classification systems
- 3 species of Iris flowers from the Gaspé penninsula

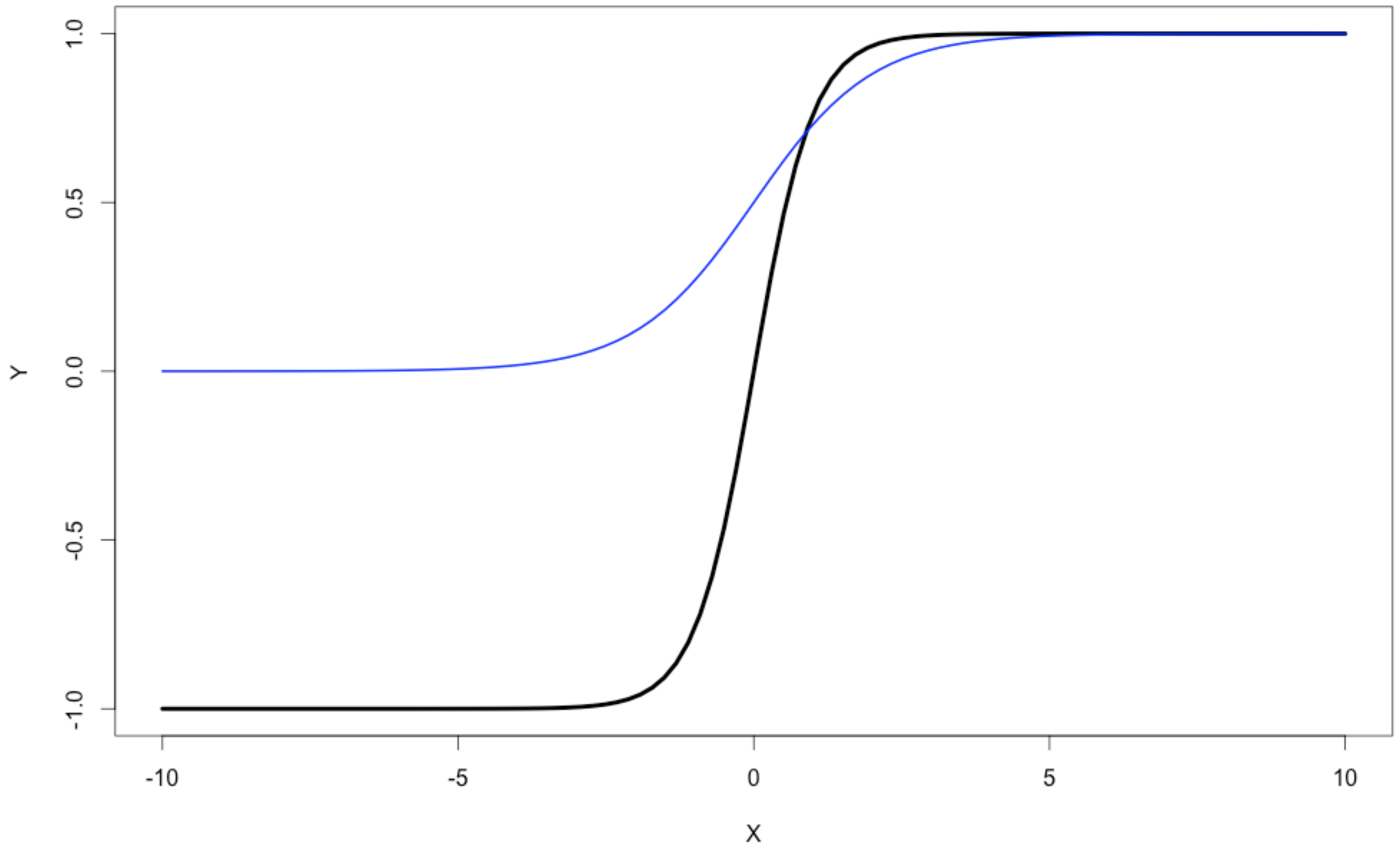


- with 2 input covariates (**x = {A,B}**) and n=3 neurons the output function o(**x**) is:

$$o(\mathbf{x}) = f\left(w_0 + \sum_{i=1}^{n} w_i x_i\right)$$

$$o(\mathbf{x}) = f\left(w_0 + \sum_{i=1}^{n} w_i x_i\right)$$

- f is bounded, monotone, and differentiable.
- Often logistic: $$f(u) = \frac{1}{1+e^{-u}}$$
- or Hyperbolic Tangent: $$tanh(u) = \frac{e^{2u}-1}{e^{2u}+1}$$



Multiple layers

- Often we build a hierarchical model:

$$o(\mathbf{x}) = f\left(w_0 + \sum_{j=1}^{J} w_j * f\left(w_{0,j} + \sum_{i=1}^{n} w_{i,j} x_{i,j}\right)\right)$$

- data —> many parallel neurons —> many parallel neurons —> …—>predictions

- In the simplest case the inner **f** is the single internal layer and the outer **f** is the output node

- library(neuralnet) uses the same **f** everywhere

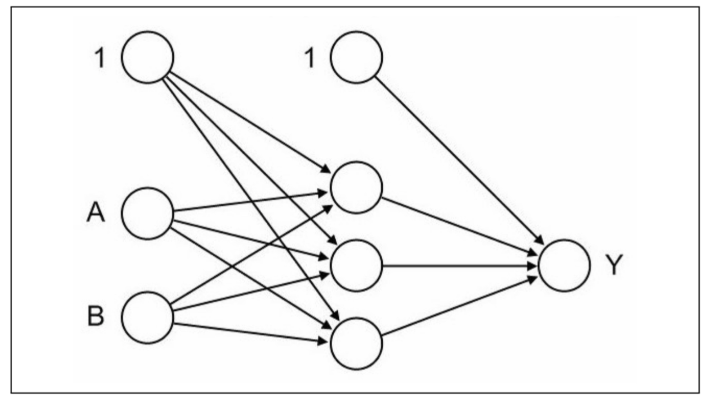


Figure 1: Example of a neural network with two input neurons (A and B), one output neuron (Y) and one hidden layer consisting of three hidden neurons.

- Better tools exist if you don't use R

Günther, Frauke, and Stefan Fritsch. "Neuralnet: Training of Neural Networks." *The R Journal* 2, no. 1 (2010): 30-38.

Models

$$E = \frac{1}{2}\sum_{l=1}^{L}\sum_{h=1}^{H}(o_{l,h} - y_{l,h})^2$$

$$o(\mathbf{x}) = f\left(w_0 + \sum_{i=1}^{n} w_i x_i\right)$$

Evaluation function E

$$f(u) = \frac{1}{1+e^{-u}}$$

Model output o

Activation function f

- How does this relate to GLMs?

Model Training

- Fitting criteria, is often Squared Error Loss (i.e. Gaussian likelihood)

$$E = \frac{1}{2}\sum_{l=1}^{L}\sum_{h=1}^{H}(o_{l,h} - y_{l,h})^2$$

- Or Cross Entropy (i.e. log Binomial likelihood):

$$E = -\sum_{l=1}^{L}\sum_{h=1}^{H}\left[y_{l,h}log(o_{l,h}) + (1 - y_{l,h})log(1 - o_{l,h})\right]$$

- for observation *l* at output node *h*

Model Training

- Optimization is usually gradient based.

$$\frac{\partial E}{\partial w} = 0|_{w=\hat{w}}$$

- Both **E** and **f** are differentiable, so gradients are analytic and often auto-differentiated (call this use of the chain rule *back propagation*)

$$\frac{\partial E}{\partial w} = \frac{dE}{do}\frac{do}{df}\frac{df}{dw}$$

- Often use CG variant or (random) subsets of dimensions to optimize at a time

Model Training

- Usually BFGS, CG, … define a step based on curvature.

- Back propagation uses learning rate **η**

$$w_k^{(t+1)} = w_k^{(t)} - \eta_k^{(t)} \left(\frac{\partial E^{(t)}}{\partial w_k^{(t)}} \right)$$

- iteration **t** and weight **k**

Improvements

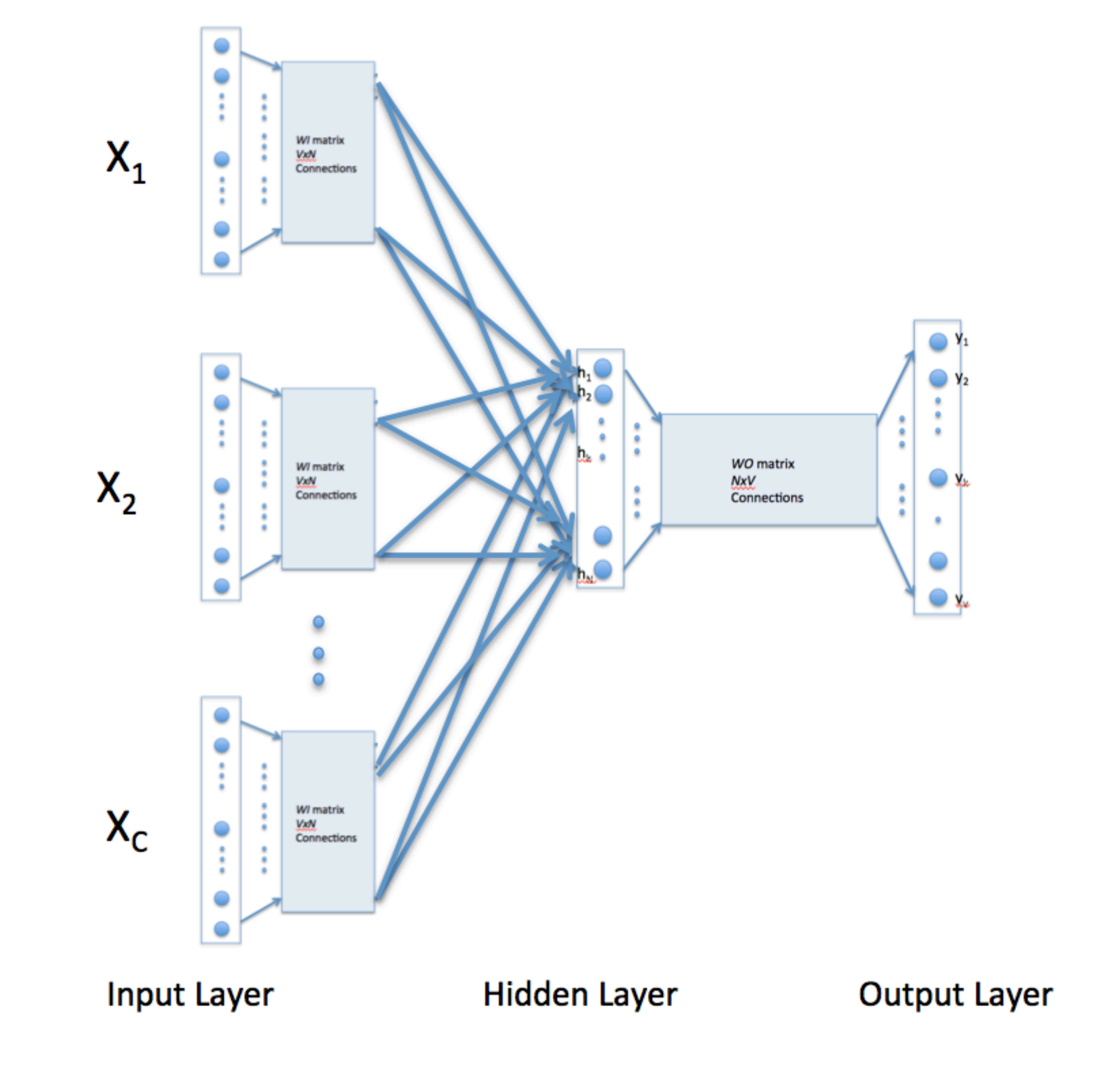
- Shutting off some weights via thresholding: sparse auto-encoders
- Imposing known structure: Convolution Neural Nets
- (much) better software

Real ML software

- Tensorflow, Theano, H2O,Caffee…, etc subdivide the neural net pieces to different GPU cores.
- Model Evaluation and optimization occur very quickly using parallel disjoint model segments and gradients.
- Several orders of magnitude speedup over R is typical

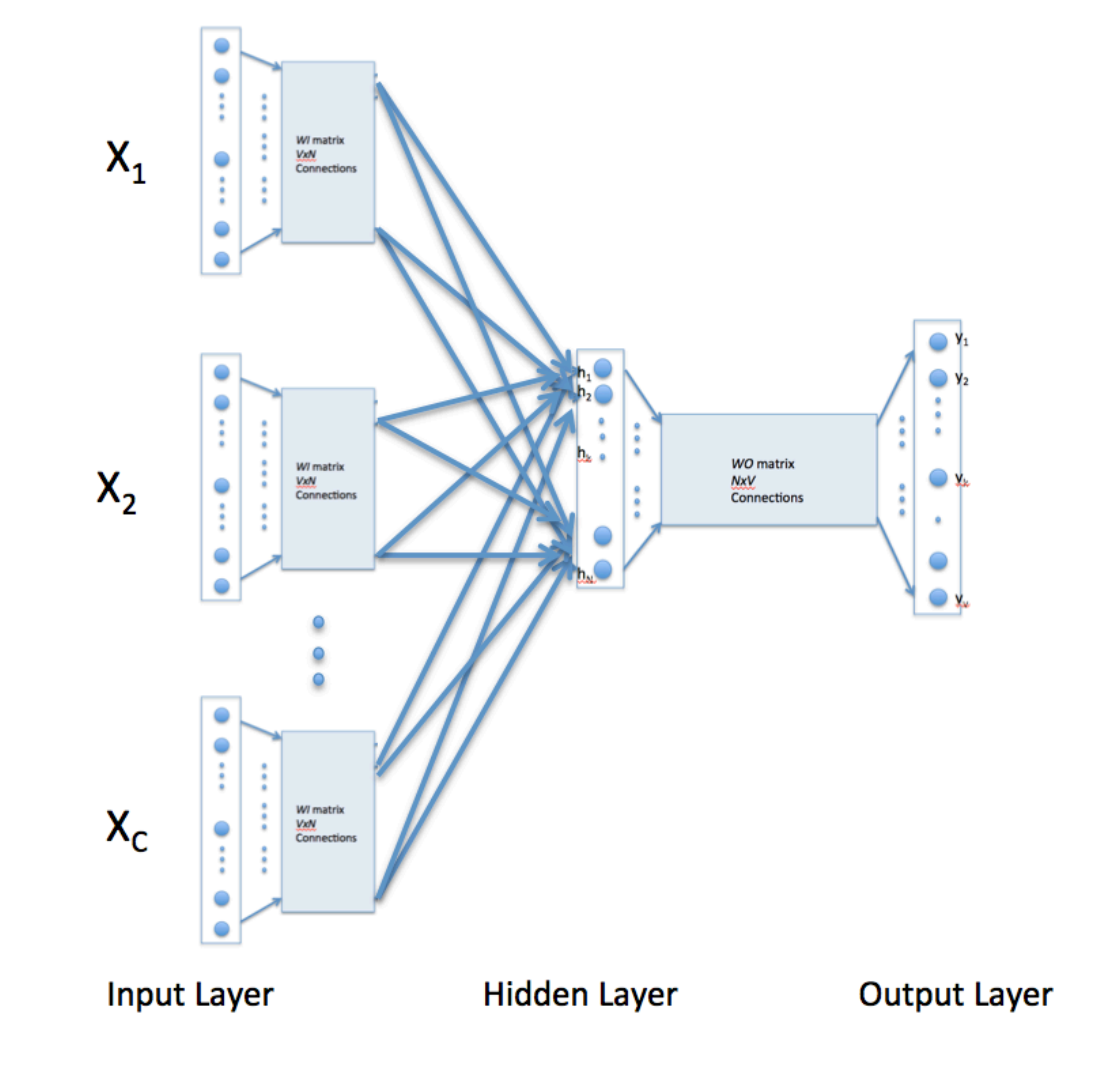
Word2Vec

- Goal: predict missing word(t)
- Input: {word(t-1), word(t-2), word(t+1), word(t+2)}
- aka: "Continuous Bag of Words" model for predicting centre word from context; BUT word order does not matter



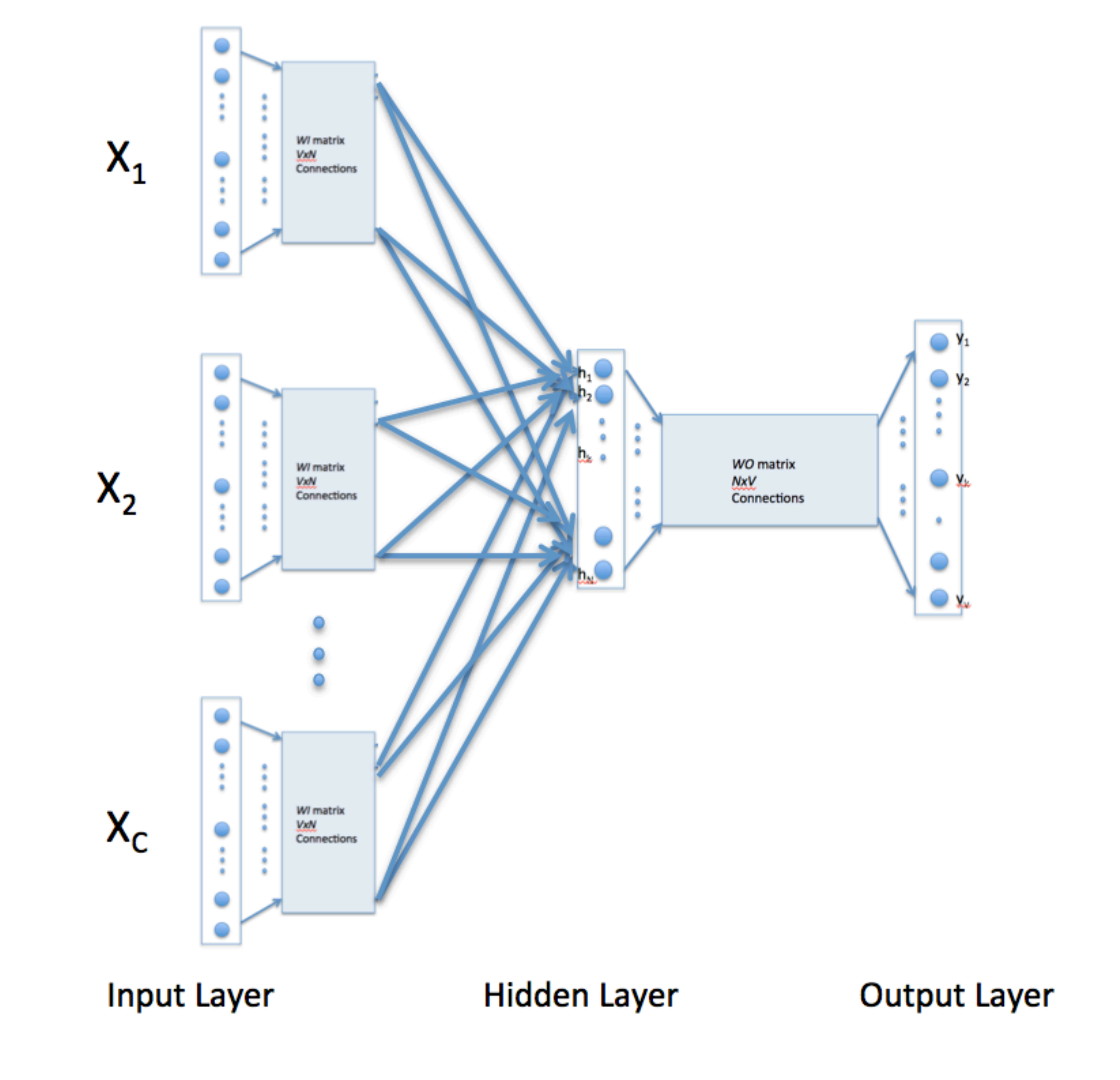
Word2Vec

- Goal: predict missing word(t)
- Input: {word(t-1), word(t-2), word(t+1), word(t+2)}
- Hidden layer: numerically combines weights and one-hot-encoded covariates



Word2Vec

- Goal: predict missing word(t)
- Input: {word(t-1), word(t-2), word(t+1), word(t+2)}
- Hidden layer: numerically combines weights and one-hot-encoded covariates
- Output: best word(t)
- Note input dimension and output dimension are the same. Internal dimension (hidden) determines the dimension of the embedding space



Word2Vec

- library(devtools)
- library(httr)
- library(tm)
- install_github("bmschmidt/wordVectors") # yup install from GitHub
- library(wordVectors)
- vignette("introduction", "wordVectors")

- Main tool is:

- train_word2vec(InputFileName,OutputFileName,vectors=LatentDimension,threads=CPUCores,window=ContextWindow,iter=ObviouslyMoreIsBetterButSlower)

- train_word2vec("cookbooks.txt","cookbook_vectors.bin",vectors=200,threads=4,window=12,iter=5)