

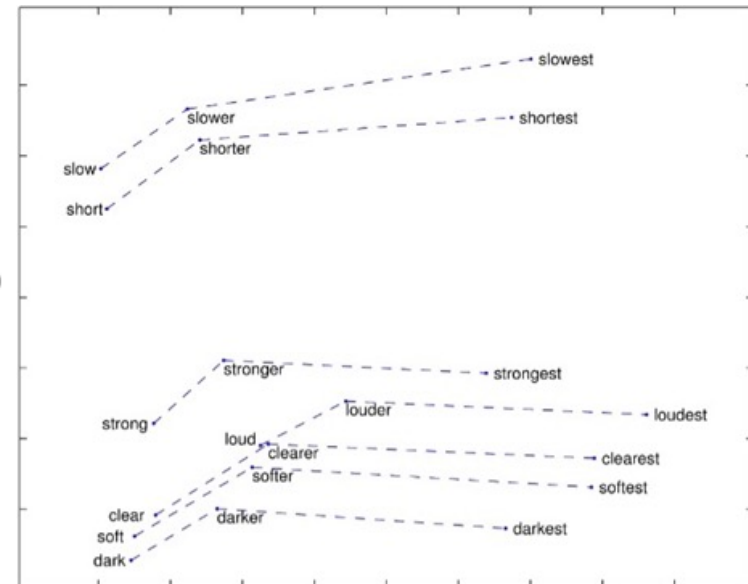
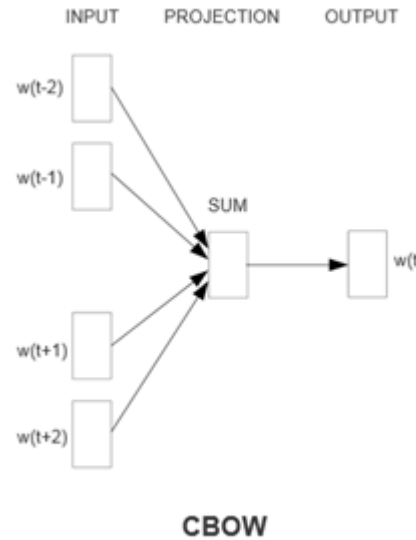
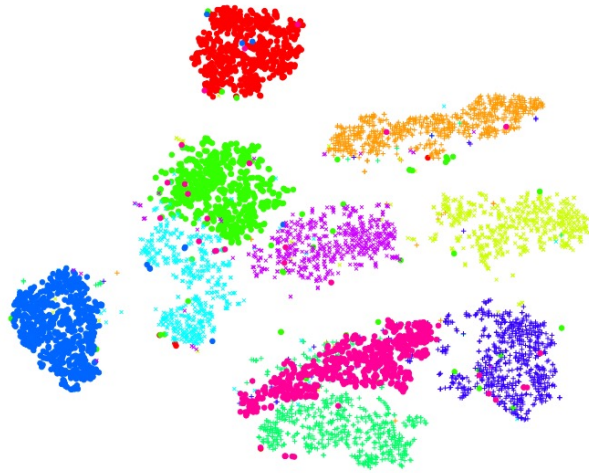
# CPSC 340: Machine Learning and Data Mining

Neural Networks

Fall 2022

# Last Time: Multi-Dimensional Scaling

- Multi-dimensional scaling (MDS):
  - Non-parametric latent-factor model: directly optimizes the  $z_i$ .
  - T-SNE tends to visualize clusters and manifold structures.
  - Word2vec gives continuous alternative to bag of words.



# End of Part 4: Key Concepts

- We discussed **linear latent-factor models**:

$$\begin{aligned} f(W, Z) &= \sum_{i=1}^n \sum_{j=1}^d (\langle w_j, z_i \rangle - x_{ij})^2 \\ &= \sum_{i=1}^n \|W^T z_i - x_i\|^2 \\ &= \|Z W - X\|_F^2 \end{aligned}$$

- Represent 'X' as linear combination of **latent factors 'w<sub>c</sub>'**.
  - **Latent features 'z<sub>i</sub>'** give a lower-dimensional version of each 'x<sub>i</sub>'.
  - When k=1, finds **direction that minimizes squared orthogonal distance**.
- Applications:
  - Outlier detection, dimensionality reduction, data compression, features for linear models, visualization, factor discovery, filling in missing entries.

# End of Part 4: Key Concepts

- **Principal component analysis (PCA):**
  - Often uses **orthogonal factors** and fits them **sequentially** (via **SVD**).
  - Or uses non-orthogonal factors and fits with SGD.
- **Generalizations of PCA** using ideas from linear models:
  - Binary PCA, robust PCA, regularized PCA, sparse PCA, non-linear PCA.]
- **Recommender systems:**
  - “Content-based filtering” is usually supervised learning approach.
  - **Collaborative-filtering** only uses ratings.
- Matrix factorization approach to collaborative filtering.
  - Fits **regularized PCA to available entries** in matrix, to “fill in” other entries.

# End of Part 4: Key Concepts

- We discussed **multi-dimensional scaling (MDS)**:
  - **Non-parametric** method for high-dimensional **data visualization**.
  - Tries to match distance/similarity in high-/low-dimensions.
    - “Gradient descent on scatterplot points”.
- Main **challenge in MDS methods is “crowding”** effect:
  - Methods focus on large distances and lose local structure.
- We discussed **t-SNE**:
  - MDS focusing on neighbour distances and not large distances.
- **Word2vec** is a recent MDS method giving better “word features”.

Next Topic: Neural Networks

# Neural Network History

- Popularity of neural networks has come in waves over the years.
  - Currently, it is **one of the hottest topics in science**.
- Recent popularity due to **unprecedented performance** on some difficult tasks.
  - Speech recognition.
  - Computer vision.
  - Natural language processing.
- These are mainly due to big datasets, deep models, and tons of computation.
  - Plus tweaks to classic models and focus on structures of networks (CNNs, LSTMs).
- For a NY Times article discussing some of the history/successes/issues, see:
  - <https://mobile.nytimes.com/2016/12/14/magazine/the-great-ai-awakening.html>

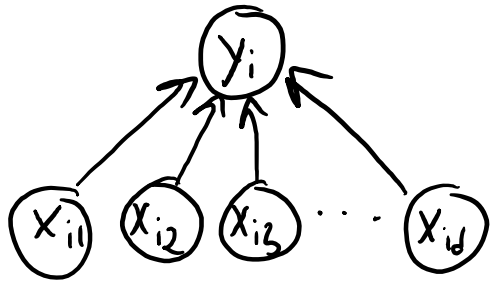
# Neural Networks: Motivation

- Many domains **require non-linear transforms** of the features.
  - But, it **may be obvious which transform** to use.
- **Neural network** models try to **learn good transformations**.
  - Optimize the “parameters of the features”.
    - And choose a class of features that have the ability to represent many functions.
- We will first discuss the special case of “one hidden layer”.
  - Then we will move onto “deep learning” with uses multiple layers.

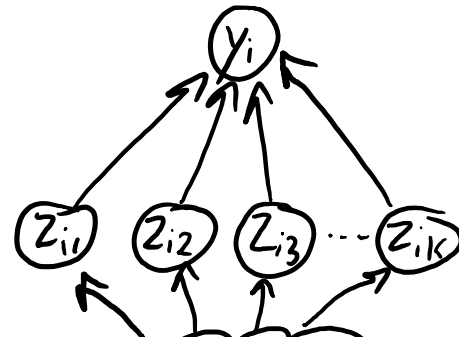


# A Graphical Summary of CPSC 340 Parts 1-5

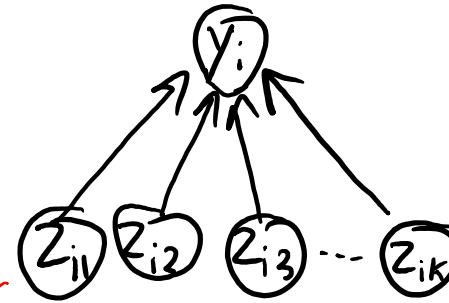
Part 1: "I have features  $x_i$ "



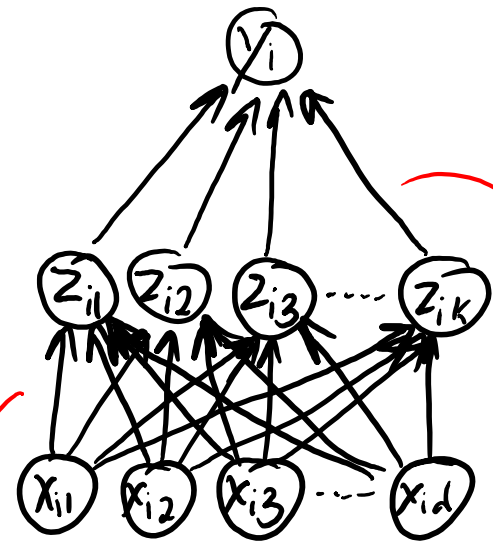
Part 3: change of basis



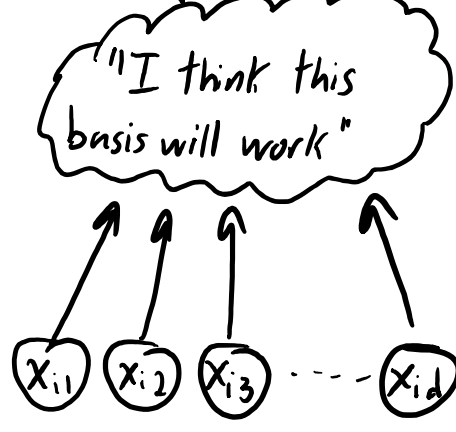
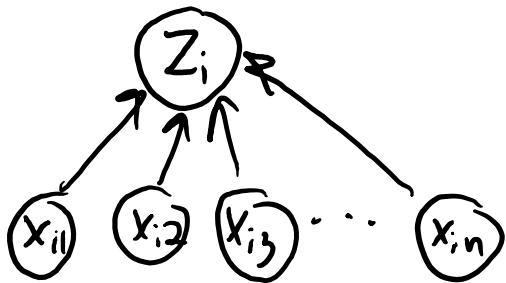
Part 4: basis from latent-factor model



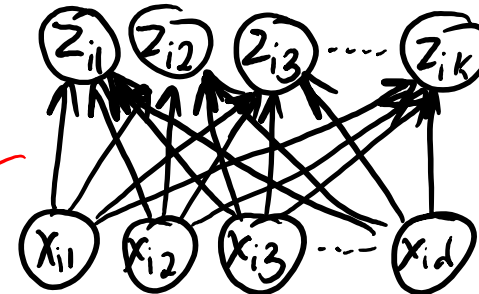
Part 5: Neural networks



Part 2: "What is the group of  $x_i$ ?"



"PCA will give me good features"



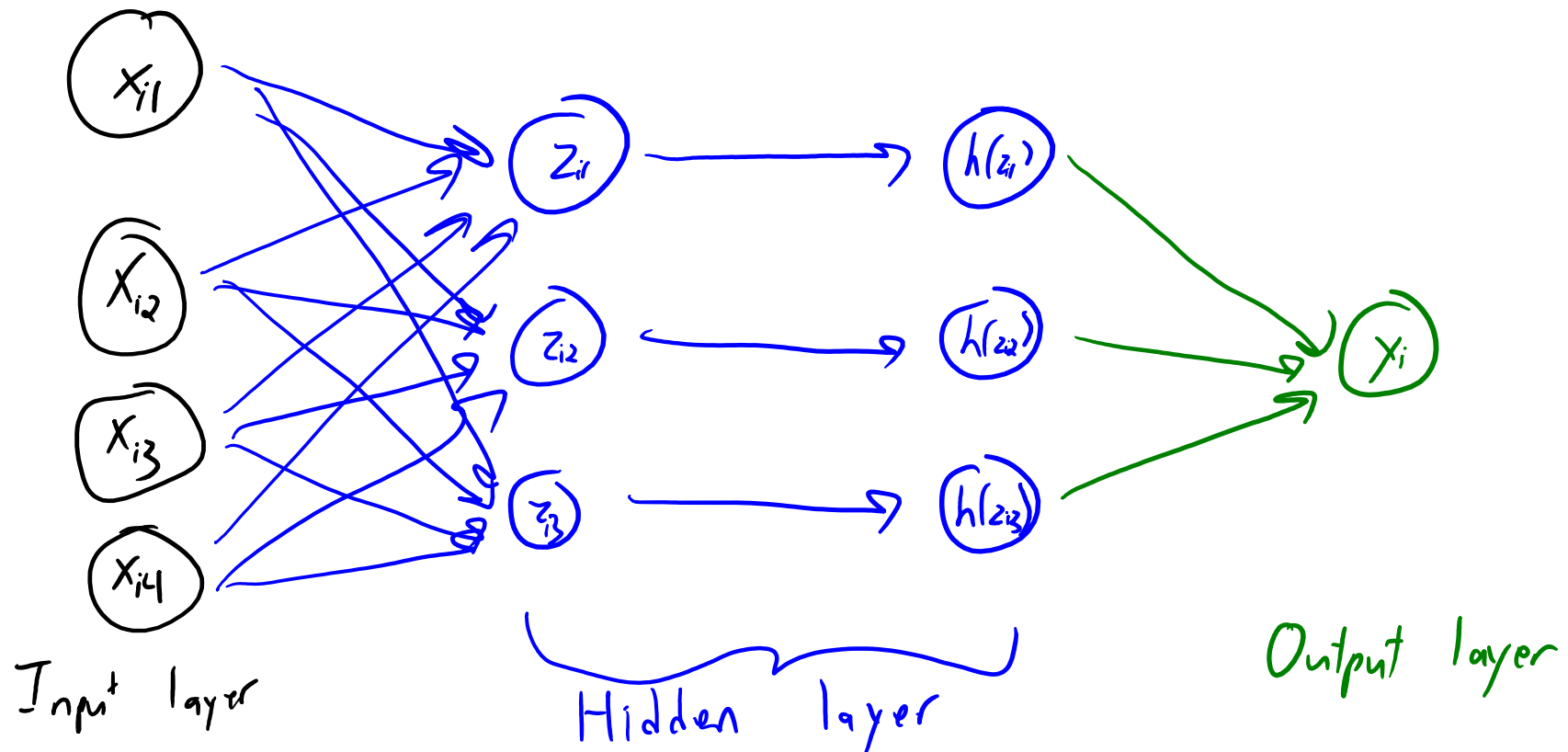
"What are the 'parts' of  $x_i$ ?"

Trained separately

Learn features and classifier at the same time.

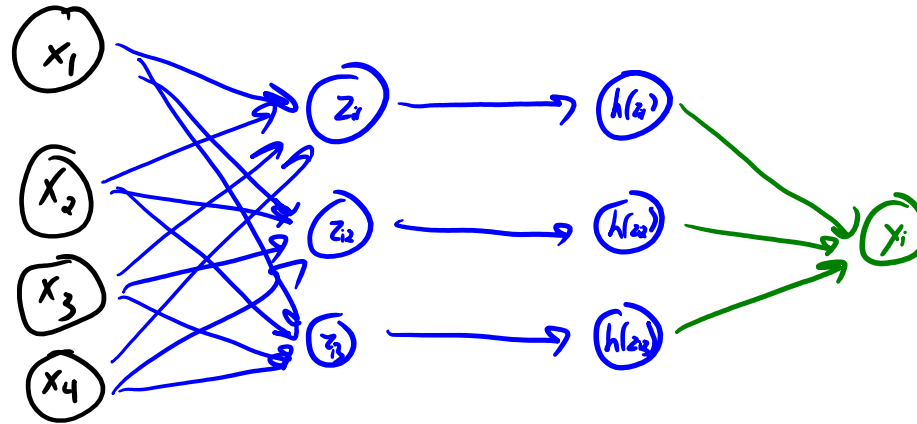
# Neural Network with One Hidden Layer

- Classic **neural network** structure with **one hidden layer**:



# Neural Network with One Hidden Layer

- As a picture:



- As a function:

$$\hat{y}_i = v^T h(W x_i)$$

Linear combination of "activations"

Non-linear transformation of each  $z_i$

" $z_i$ ": linear combination of input

$$Z = \begin{bmatrix} \text{---} z_1^T \text{---} \\ \text{---} z_2^T \text{---} \\ \vdots \\ \text{---} z_n^T \text{---} \end{bmatrix}$$

$n \times k$

# Neural Network with One Hidden Layer

- As a function:

$$\hat{y}_i = v^T h(W x_i)$$

Linear combination of "activations"

Non-linear transformation of each  $z_i$

" $z_i$ ": linear combination of input

- Parameters:** the "k times d" matrix "**W**", and length-k vector "**v**".
  - Using 'k' as number of "**hidden units**", the dimension are:

$$W = \begin{bmatrix} \text{---} & w_1^T & \text{---} \\ \text{---} & w_2^T & \text{---} \\ \vdots & \vdots & \vdots \\ \text{---} & w_k^T & \text{---} \end{bmatrix}$$

$k \times d$

$$v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_k \end{bmatrix}$$

$k \times 1$

# Neural Network with One Hidden Layer

- As a function:

$$\hat{y}_i = v^T h(W x_i)$$

Linear combination of "activations"

Non-linear transformation of each  $z_i$

" $z_i$ ": linear combination of input

- Linear transformation  $z_i = Wx_i$  is like doing PCA.
  - Mixes together the features in a way that we learn.
- Non-linear transform 'h' is often sigmoid applied element-wise.
  - Without a non-linear transformation it degenerates to a linear model:
    - $\hat{y}_i = v^T(Wx_i) = (v^TW)x_i = w^Tx_i$  (if we set 'w' using  $w = W^Tv$ ).

# Neural Network with One Hidden Layer

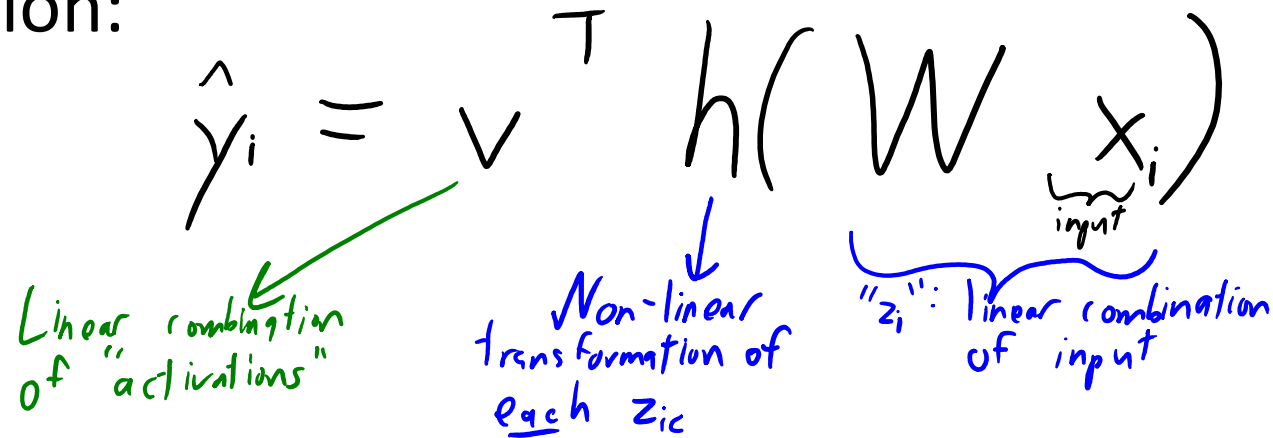
- As a function:

$$\hat{y}_i = v^T h(W x_i)$$

Linear combination of "activations"

Non-linear transformation of each  $z_i$

" $z_i$ ": linear combination of input

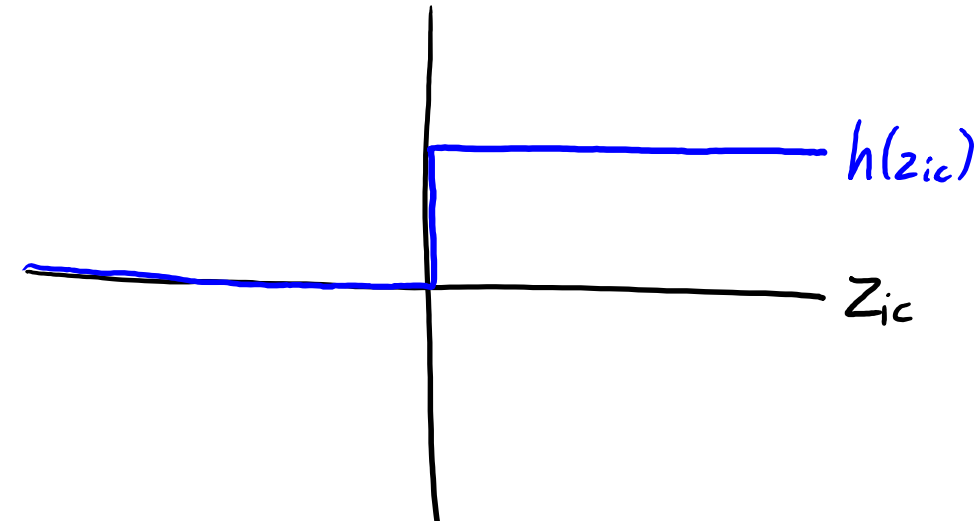
The image shows the handwritten equation  $\hat{y}_i = v^T h(W x_i)$ . A green arrow points from the text "Linear combination of 'activations'" to the  $v^T$  term. A blue arrow points from the text "Non-linear transformation of each  $z_i$ " to the  $h(\cdot)$  function. A blue bracket under the  $x_i$  term is labeled "input", and another blue bracket under the  $W x_i$  term is labeled " $z_i$ : linear combination of input".

- **Second linear transformation**  $v^T h(z_i)$  gives final value.
  - This is like using a **linear model with non-linear feature** transformations.
    - But in this case we **learned the features**.
- Cost of computing  $\hat{y}_i$  above is  **$O(kd)$** .
  - $O(kd)$  to compute  $Wx_i$ ,  $O(k)$  to apply 'h', then  $O(k)$  to multiply by 'v'.

# Why Sigmoid as Non-Linear Transform?

- Consider setting 'h' to define **binary features**  $z_i$  using:

$$h(z_{ic}) = \begin{cases} 1 & \text{if } z_{ic} \geq 0 \\ 0 & \text{if } z_{ic} < 0 \end{cases}$$



- Each  $h(z_i)$  can be viewed as binary feature.
  - “You either have this ‘part’ or you don’t have it.”
- We can make  $2^k$  objects by all the possible “part combinations”.

Motivation: Pixels vs. Parts

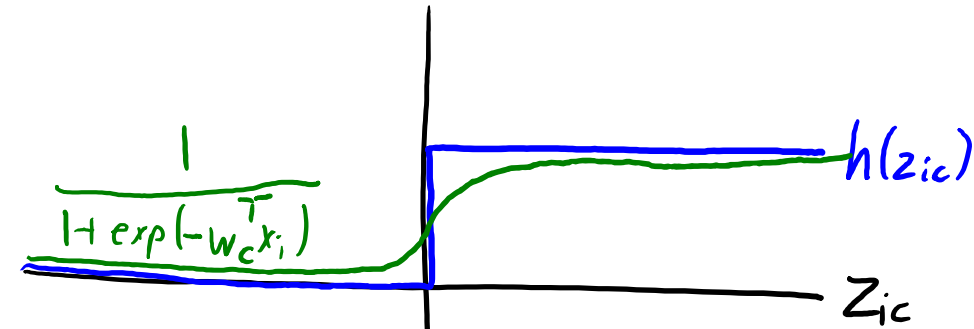
- We could represent other digits as different combinations of “parts”:

3	=	1	+	1	+	1	+	1	+	1	+	0	+	0
5	=	1	+	0	+	1	+	1	+	1	+	0	+	1
8	=	1	+	1	+	1	+	1	+	1	+	1	+	1

# Why Sigmoid as Non-Linear Transform?

- Consider setting 'h' to define **binary features**  $z_i$  using:

$$h(z_{ic}) = \begin{cases} 1 & \text{if } z_{ic} \geq 0 \\ 0 & \text{if } z_{ic} < 0 \end{cases}$$



- Each  $h(z_i)$  can be viewed as binary feature.
  - “You either have this ‘part’ or you don’t have it.”
- But this is hard to optimize (**non-differentiable/discontinuous**).
- **Sigmoid is a smooth approximation** to these binary features.
  - Allows you to train the model using gradient descent or SGD.



# Universal Approximation with One Hidden Layer

- Classic choice of “activation” function ‘h’ is the sigmoid function.
- With enough hidden “units”, this is a “universal approximator”.
  - Any continuous function can be approximated arbitrarily well (on bounded domain).
- But this result is for a non-parametric setting of the parameters:
  - The number of hidden “units” must be a function of ‘n’.
  - A fixed-size network is not a universal approximator.
- Other universal approximators (always non-parametric):
  - K-nearest neighbours.
    - Need to have ‘k’ depending on ‘n’.
  - Linear models with polynomial non-linear features transformations.
    - Degree of polynomial depends on ‘n’.
  - Linear models with Gaussian RBFs as non-linear features transformations or kernels.
    - With RBF centered on each  $x^i$ .

# Adding Bias Variables

- Recall fitting linear models with a **bias variable** (so  $\hat{y}_i \neq 0$  when  $x_i=0$ ).

$$\hat{y}_i = \sum_{j=1}^d w_j x_{ij} + \beta$$

- We often implement this by **adding a column of ones to X**.
- In neural networks we often include **biases on each  $z_{ic}$** :

$$\hat{y}_i = \sum_{c=1}^K v_c h(w_c^T x_i + \beta_c)$$

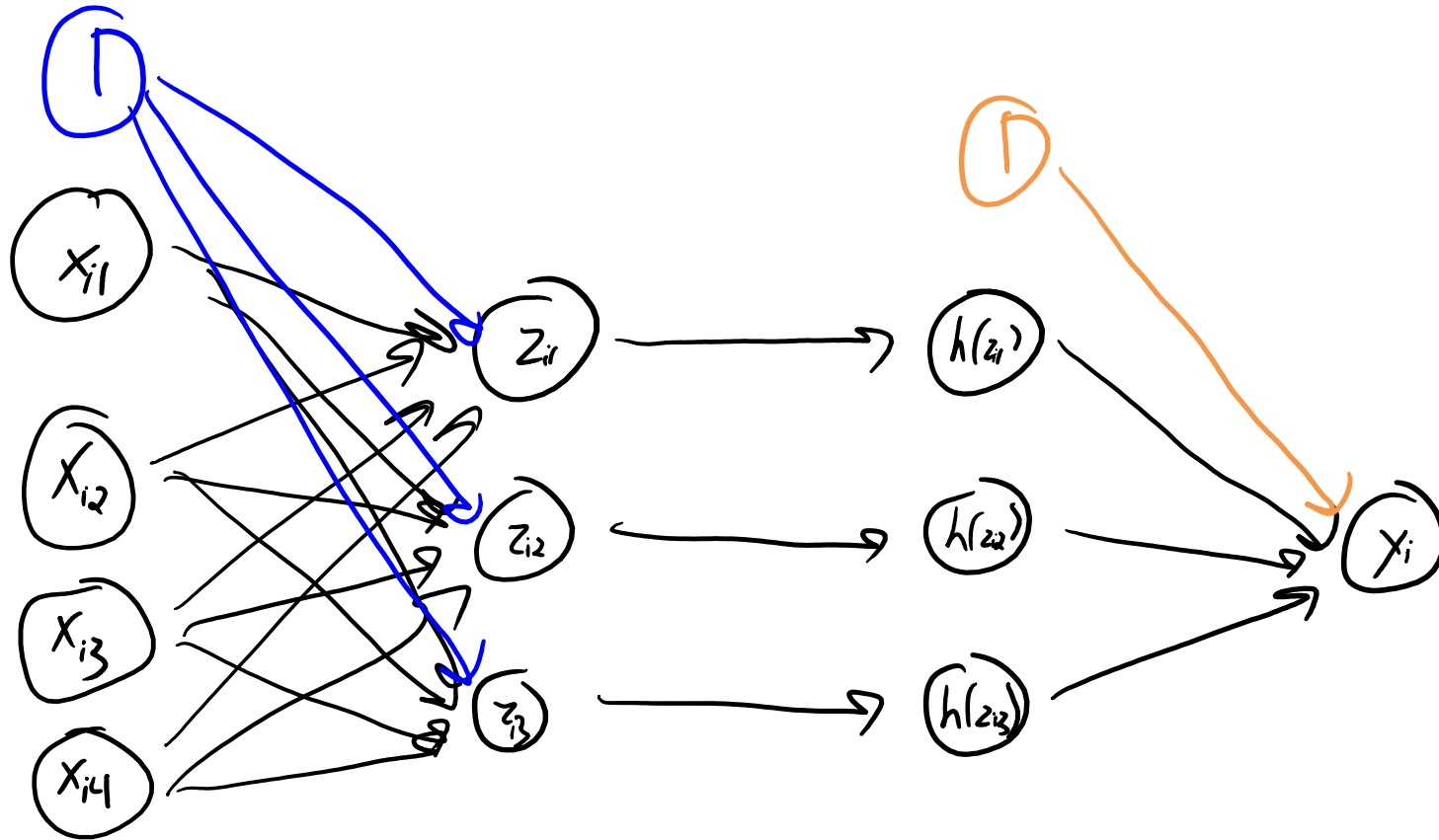
- As before, we could implement this by **adding a column of ones to X**.
- We often also want a **bias on the output**:

$$\hat{y}_i = \sum_{c=1}^K v_c h(w_c^T x_i + \beta_c) + \beta$$

- For sigmoid 'h', you could equivalently fix **one row of W to be 0**.
  - Since  $h(0)$  is a constant.

# Adding Bias Variables

$$\hat{y}_i = \sum_{c=1}^K v_c h(w_c^T x_i + \beta_c) + \beta$$



# Regression and Binary Classification

- For **regression** problems, our prediction (ignoring biases) is:

$$\hat{y}_i = v^T h(W x_i)$$

- And we might train to minimize the **squared residual**:

$$f(W, v) = \frac{1}{2} \sum_{i=1}^n (\hat{y}_i - y_i)^2 = \frac{1}{2} \sum_{i=1}^n (v^T h(W x_i) - y_i)^2$$

- For **binary classification**, our prediction (ignoring biases) is:

$$\text{sign}(v^T h(W x_i)) \quad \text{or} \quad p(y_i | W, v, x_i) = \frac{1}{1 + \exp(-y_i v^T h(W x_i))}$$

Use a Sigmoid on output to get a probability

- And we might train to minimize the **logistic loss**:

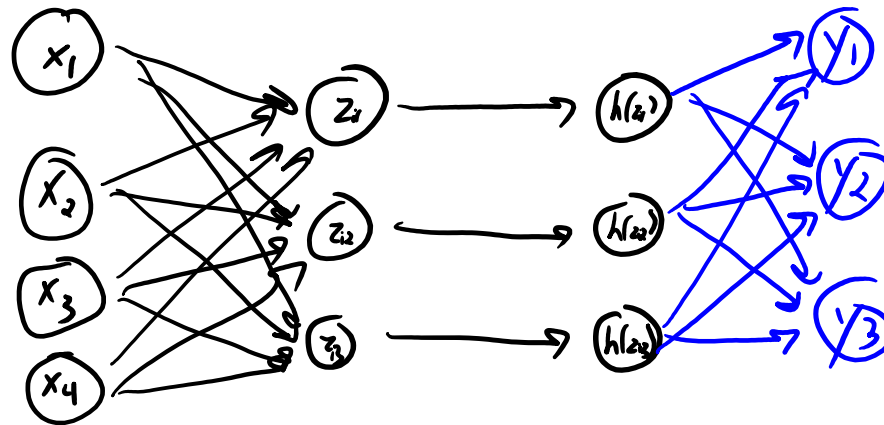
$$f(W, v) = \sum_{i=1}^n \log(1 + \exp(-y_i \hat{y}_i)) = \sum_{i=1}^n \log(1 + \exp(-y_i v^T h(W x_i)))$$

- This is like **logistic regression with learned features**.

# Neural Network for Multi-Class Classification

- Multi-class classification with a neural network:

- Input is connected to a hidden layer (same as regression and binary case).
- Hidden layer is connected to multiple output units (one for each label.).



$$\hat{y}_1 = v_1^T h(Wx)$$

$$\hat{y}_2 = v_2^T h(Wx)$$

$$\hat{y}_3 = v_3^T h(Wx)$$

Now have a matrix of parameters:

$$V = \begin{bmatrix} \text{---} v_1^T \text{---} \\ \text{---} v_2^T \text{---} \\ \vdots \\ \text{---} v_{k'}^T \text{---} \end{bmatrix}$$

$k' \times K$   
 number of classes  $\rightarrow$  number of hidden units

- We convert to probabilities for each class using softmax of the  $\hat{y}_c$  values.

$$p(y_i = c | x_i, W, V) = \frac{\exp(\hat{y}_c)}{\sum_{c'=1}^{k'} \exp(\hat{y}_{c'})}$$

- We can predict by maximizing  $p(y_i | x_i, W, V)$  over all each 'c' (one prediction across classes).
- We train by minimizing negative log of this probability (softmax loss, summed across examples).
- Notice that we changed tasks by only changing last layer (and loss function).

# Summary

- Unprecedented performance on difficult pattern recognition tasks.
- Neural networks with one hidden layer:
  - Simultaneous learn a linear model and its features  $z_i$ .
- Non-linear transform avoids degeneracy.
  - Universal approximator if size of layer grows with number of examples 'n'.
- Bias variables added to each layer.
- Outputting probabilities and training with SGD.
- Next time: neural networks overfit less with more parameters?

# Is Training Neural Networks Scary?

- Learning:

- For binary classification, the NLL under the sigmoid likelihood is:

$$f(W, v) = \sum_{i=1}^n \underbrace{\log(1 + \exp(-y_i v^T h(Wx_i)))}_{f_i}$$

*loss function on example  $i$*

- With 'W' fixed this is convex, but with both 'W' and 'v' as variables it is **non-convex**.
- And finding the global optimum is **NP-hard** in general.

- Nearly-always trained with variations on **stochastic gradient descent (SGD)**.

$$\begin{aligned} W^{k+1} &= W^k - \alpha^k \nabla_W f_{i_k}(W^k, v^k) \\ v^{k+1} &= v^k - \alpha^k \nabla_v f_{i_k}(W^k, v^k) \end{aligned}$$

*$i_k$  is a training example chosen uniformly at random*

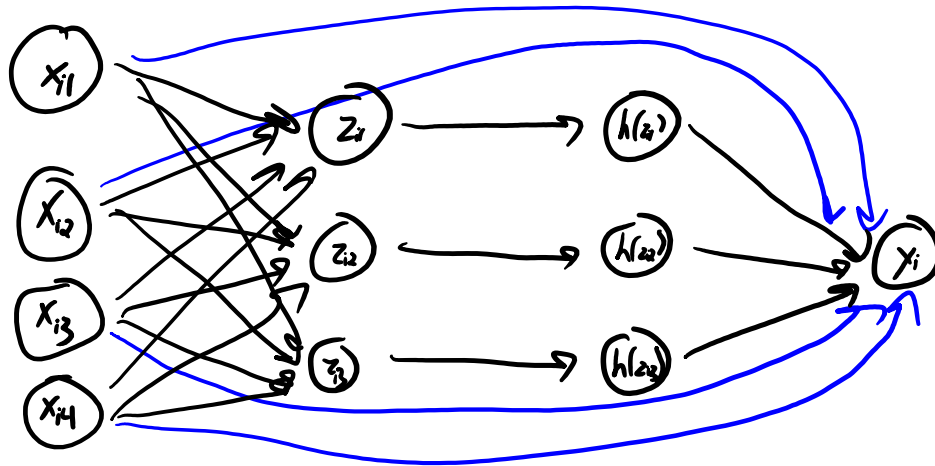
- Many variations exist (adding "momentum", AdaGrad, Adam, and so on).
- But **SGD is not guaranteed to reach a global minimum** for non-convex problems.

- Is non-convexity a big drawback compared to logistic regression?

- And if 'k' is large, is this likely to overfit?

# Neural Networks $\geq$ Logistic Regression

- Consider a neural network with one hidden layer and **connections from input to output layer**.
  - The extra connections are called “**skip**” connections.



$$\hat{y}_i = \underbrace{w^T x_i}_{\text{linear model}} + \underbrace{v^T h(Wx_i)}_{\text{neural network}}$$

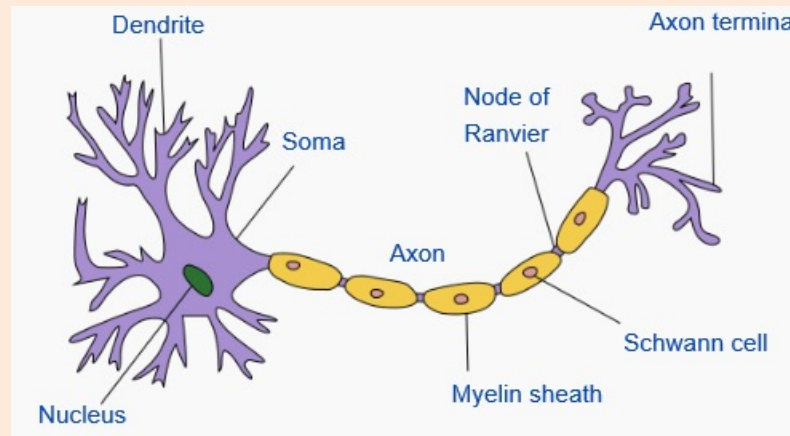
- You could first set  $v=0$ , then **optimize ‘w’ using logistic regression**.
  - This is a convex optimization problem that gives you the logistic regression model.
- You could then set ‘W’ and ‘v’ to small random values, and start SGD from the logistic regression model.
  - And if you are worried about overfitting, you could use **early stopping** based on validation set.
  - Even though this is non-convex, the neural network **can only improve on logistic regression**.
- In practice, we typically optimize everything at once (which usually works better than the above).



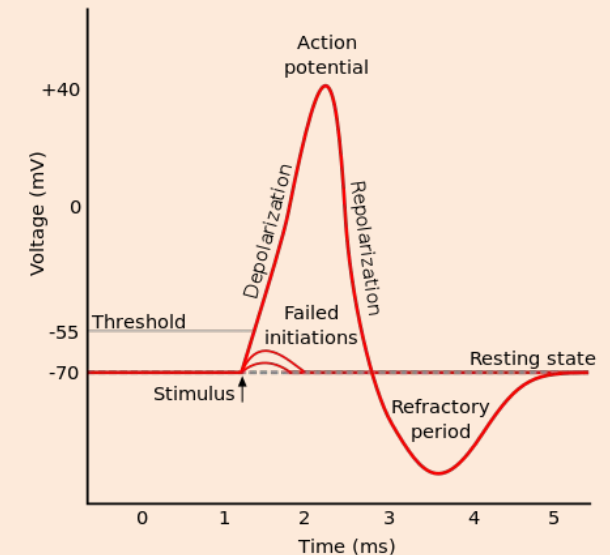
Next Topic: Biological Motivation

# Why “Neural Network”?

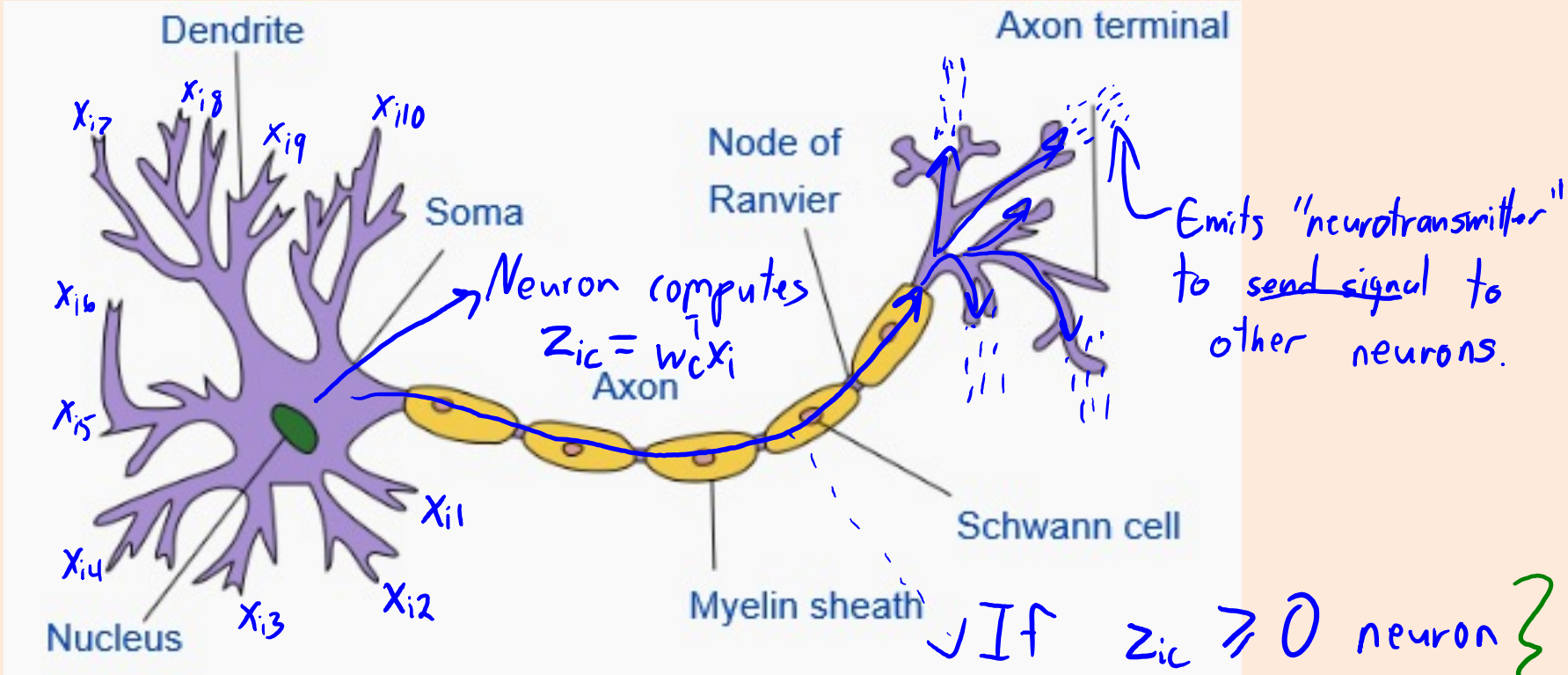
- Cartoon of “typical” neuron:



- Neuron has many “dendrites”, which take an input signal.
- Neuron has a single “axon”, which sends an output signal.
- With the right input to dendrites:
  - “Action potential” along axon (like a binary signal):

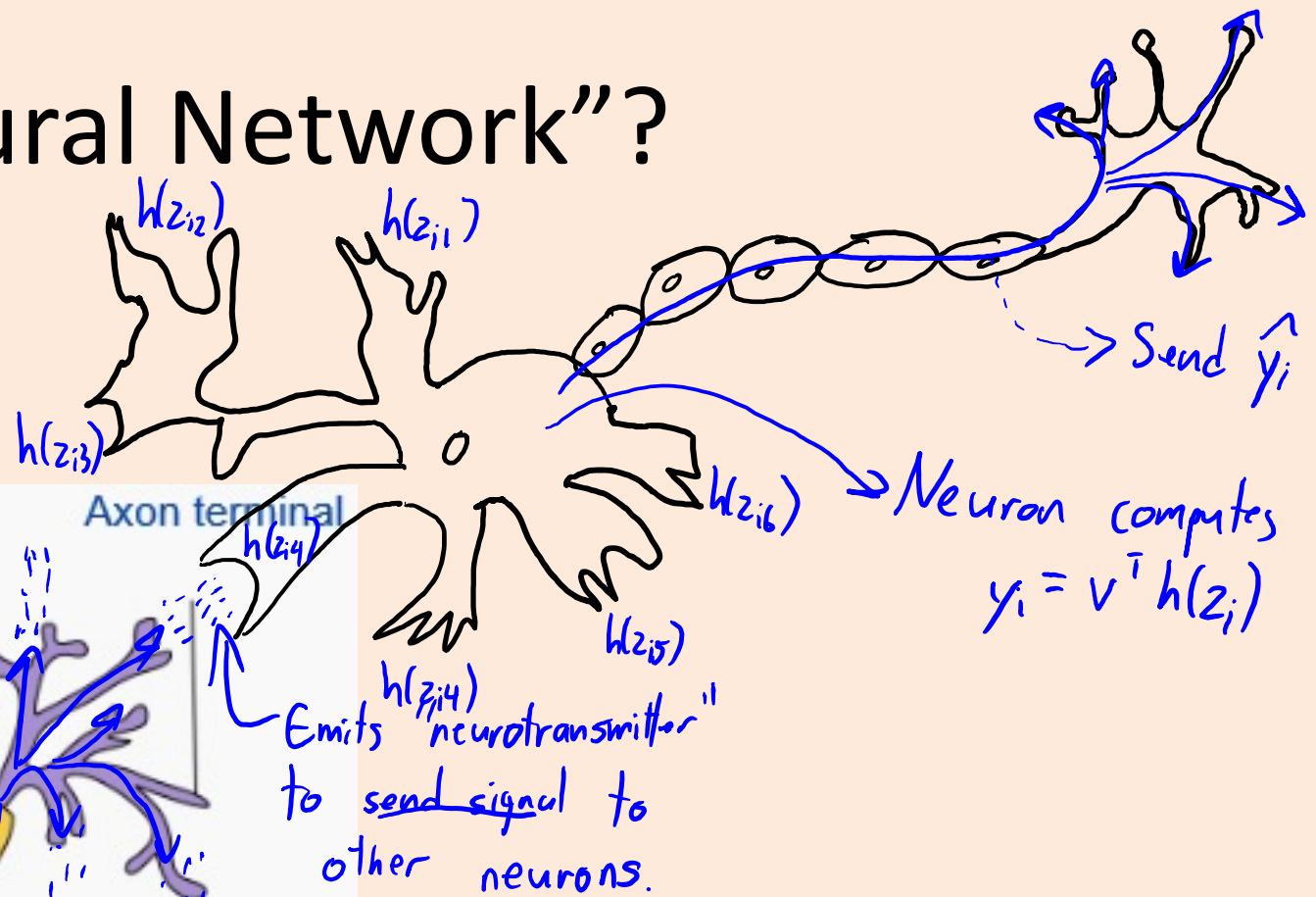
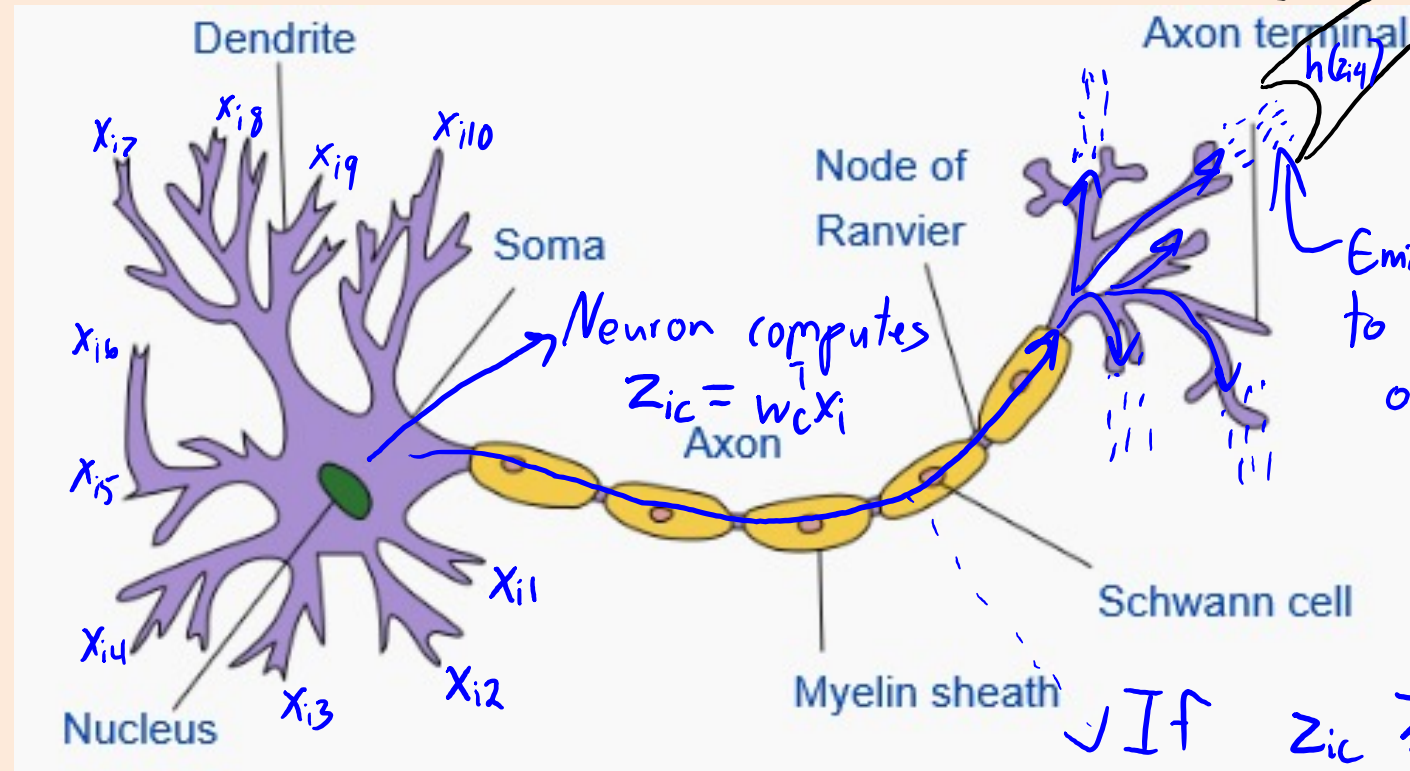


# Why "Neural Network"?



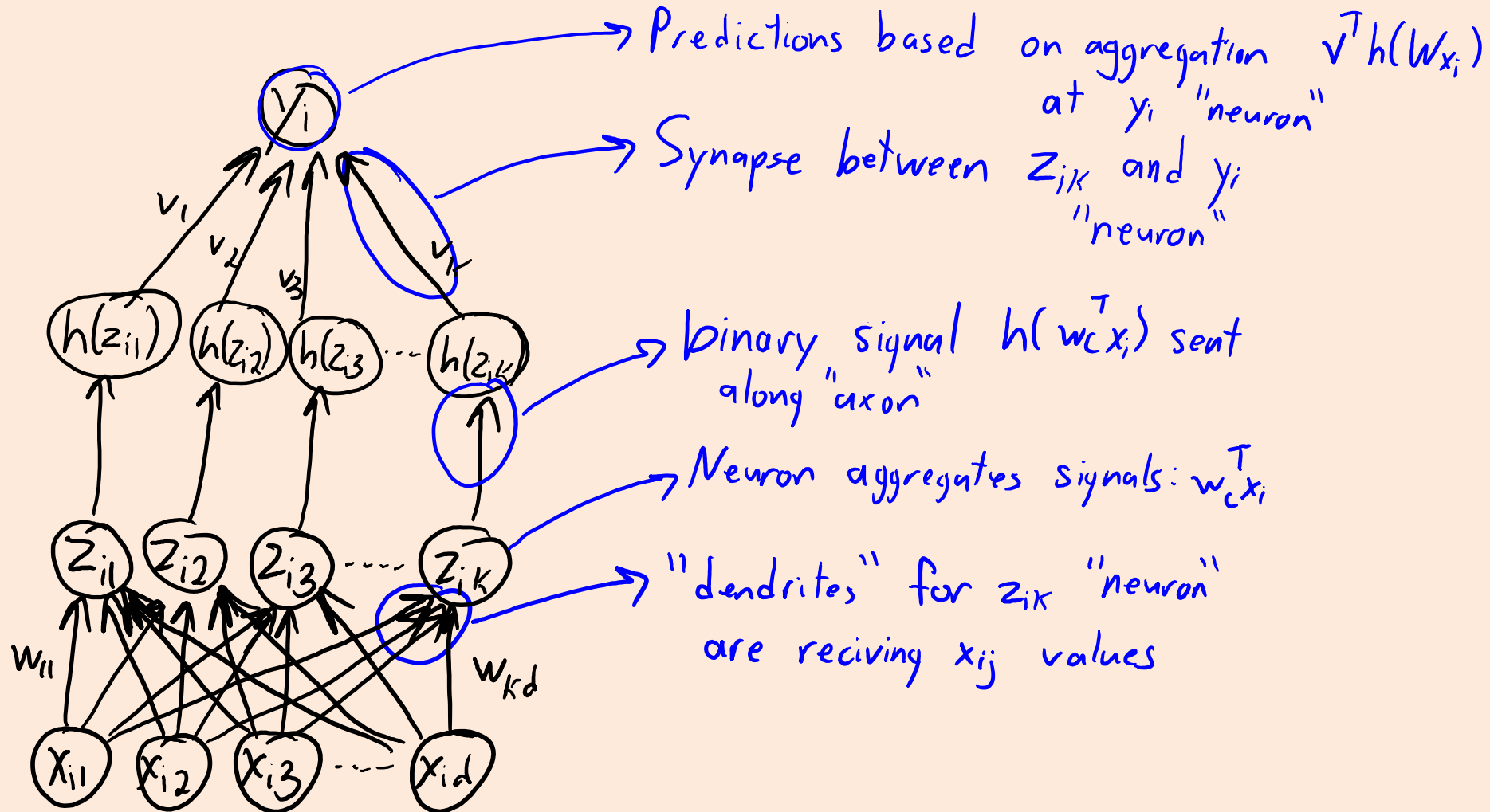
If  $z_{ic} \geq 0$  neuron } We approximate binary  
 Sends signal along axon. } signal with  $\frac{1}{1 + \exp(-z_{ic})}$

# Why "Neural Network"?



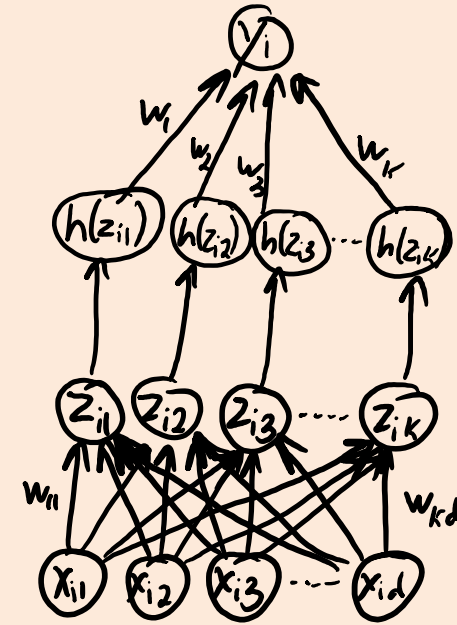
If  $z_{ic} \geq 0$  neuron } We approximate binary  
 Sends signal along axon. } signal with  $\frac{1}{1 + \exp(-z_{ic})}$

# Why "Neural Network"?



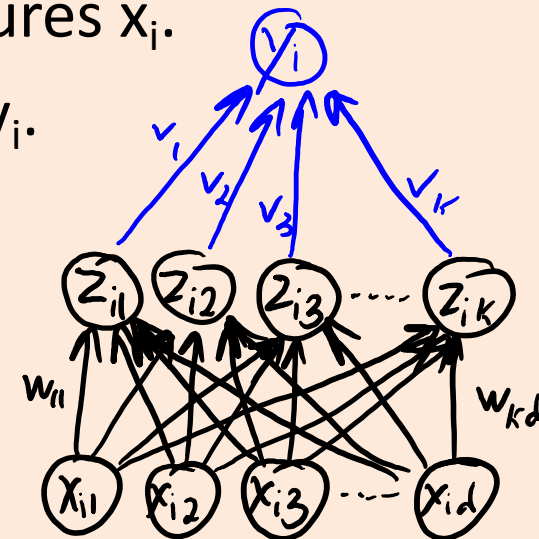
# “Artificial” Neural Nets vs. “Real” Networks Nets

- Artificial neural network:
  - $x_i$  is measurement of the world.
  - $z_i$  is internal representation of world.
  - $y_i$  is output of neuron for classification/regression.
- Real neural networks are more complicated:
  - **Timing** of action potentials seems to be important.
    - “Rate coding”: frequency of action potentials simulates continuous output.
  - Neural networks don’t reflect **sparsity** of action potentials.
  - How much computation is done **inside neuron**?
  - Brain is highly **organized** (e.g., substructures and cortical columns).
  - Connection **structure changes**.
  - **Different types** of neurotransmitters.



# Supervised Learning Roadmap

- Part 1: “Direct” **Supervised Learning**.
  - We learned parameters ‘ $w$ ’ based on the **original features  $x_i$**  and target  $y_i$ .
- Part 3: **Change of Basis**.
  - We learned parameters ‘ $v$ ’ based on a **change of basis  $z_i$**  and target  $y_i$ .
- Part 4: **Latent-Factor Models**.
  - We **learned parameters ‘ $W$ ’ for basis  $z_i$**  based on only on features  $x_i$ .
  - You can **then learn ‘ $v$ ’** based on change of basis  $z_i$  and target  $y_i$ .
- Part 5: **Neural Networks** (one hidden layer).
  - **Jointly learn ‘ $W$ ’ and ‘ $v$ ’ based on  $x_i$  and  $y_i$ .**
  - **Learn basis  $z_i$  that is good for supervised learning.**



# Why $z_i = Wx_i$ ?

- In PCA we had that the optimal  $Z = XW^T(WW^T)^{-1}$ .
- If  $W$  had normalized+orthogonal rows,  $Z = XW^T$  (since  $WW^T = I$ ).
  - So  $z_i = Wx_i$  in this normalized+orthogonal case.
- Why we would use  $z_i = Wx_i$  in neural networks?
  - We didn't enforce normalization or orthogonality.
- Well, the value  $W^T(WW^T)^{-1}$  is just “some matrix”.
  - You can think of neural networks as just **directly learning this matrix**.



# Softmax NLL vs. Cross-Entropy

- Multi-class objective often written as minimizing **cross-entropy**:

$$f(W, V) = \sum_{i=1}^n \sum_{c=1}^C I[y^i=c] (-\log p(y^i=c | X, W, V))$$

- The indicator function is **zero except for true label  $y^i$** :

$$f(W, V) = -\sum_{i=1}^n \log p(y^i | X, W, V)$$

- When we plug in the softmax likelihood, we get the **softmax NLL**.
  - So **cross-entropy is the softmax NLL** with extra terms that do nothing.
    - Cross-entropy way of writing would make more sense if training data had “soft” assignments to classes.