TEXAS
The University of Texas at Austin

I320D – Topics in Human Centered Data Science

Text Mining and NLP Essentials

Week 10: Deep Learning for NLP, Introduction to Neural Networks, Recurrent Neural Networks and Transformers

Dr. Abhijit Mishra

Week 10-11 Activities

- Project Proposals Due on Friday (**03/26/2023**):
  - Maximum 2 pages – one submission per group
  - https://utexas.instructure.com/courses/1382133/assignments/6619554?module_item_id=13585860
  - Sample project reports: Canvas->Files->Sample_project_reports

Last Week

- Unsupervised ML for NLP
  - K-means clustering
  - Topic modeling Introduction - LDA Model
- Practicum: Topic extraction from text using unsupervised methods

Week 9 Recap: Unsupervised Learning

- In unsupervised learning,
  - we have no labels
  - our goals are less concrete
- Applications in NLP
  - Analyzing Corpus: Clustering Words / Documents
  - Topic Extraction from text

Week 9 Recap: Clustering

- Divide data into K-clusters
- Data could be **words , sentences, documents**
- **Key Element:** Distance between data representations
  - E.g., Euclidean Distance, Cosine Distance
- **Data Representations: Feature vectors**
  - **E.g., Words:** Word Embeddings (e.g., GloVE)**, 1-hot vectors**
  - **Sentences:** Sentence Embeddings (e.g., averaged GloVE, BERT), N-hot, TF-TDF vectors,
  - **Documents:** Similar to sentence embeddings

Week 9 Recap: Topic Extraction using Clustering

- Discussed simple algorithm
- Step 1: Pre-process all documents first

  vocabulary = {}

  For each document in corpus:

    Remove stop words

    Lemmatize each word

    Add each word to vocabulary

Week 9 Recap: Topic Extraction using Clustering

- Step1: Pre-process all documents first
- Step2: Perform feature-extraction

```
all_features = {}
For each word in vocabulary:
    feature_vector := GloVE (word)
    all_features.add(feature_vector)
```

Week 9 Recap: Topic Extraction using Clustering

- Step1: Pre-process all documents first
- Step2: Perform feature-extraction
- Step3: Cluster words using K-means

```
Ks = [2, 3, 4, 5,⋯]
for each K  in Ks:
    clusters = KMeans (all_features, K)
    distortion = within_cluster_sum_of_pair-wise distances (clusters)
Optimal_K := K where distortion elbows
```

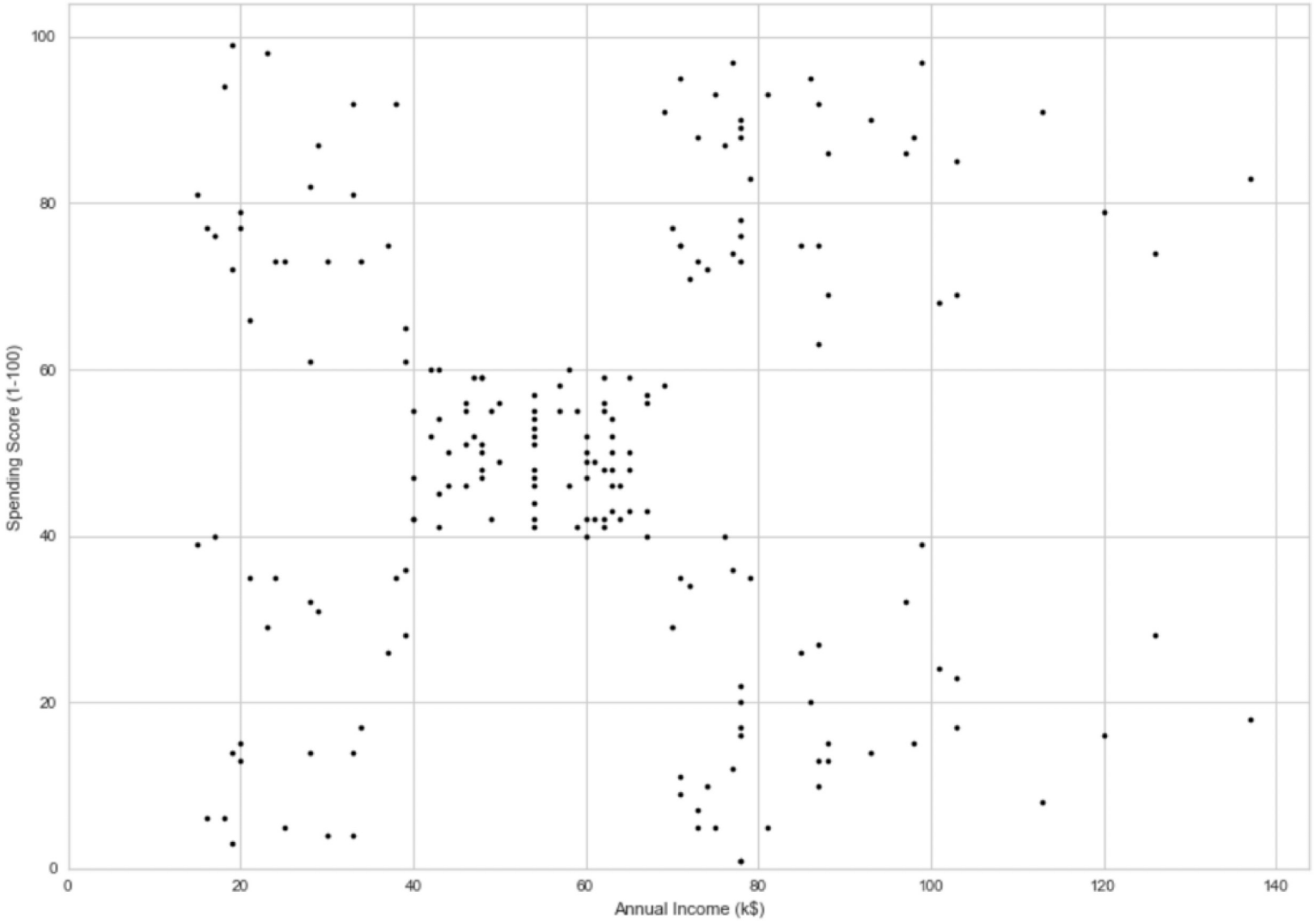
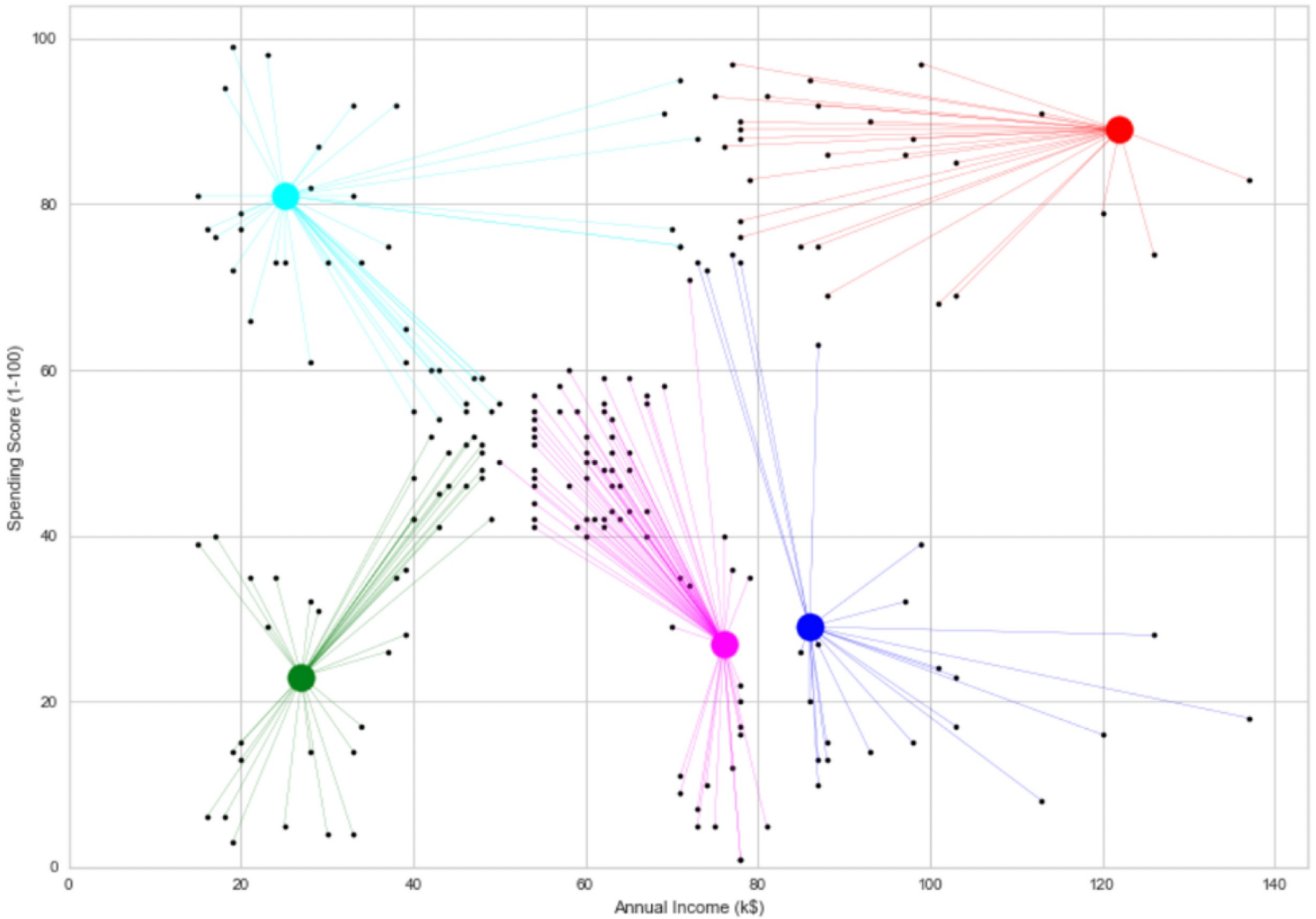
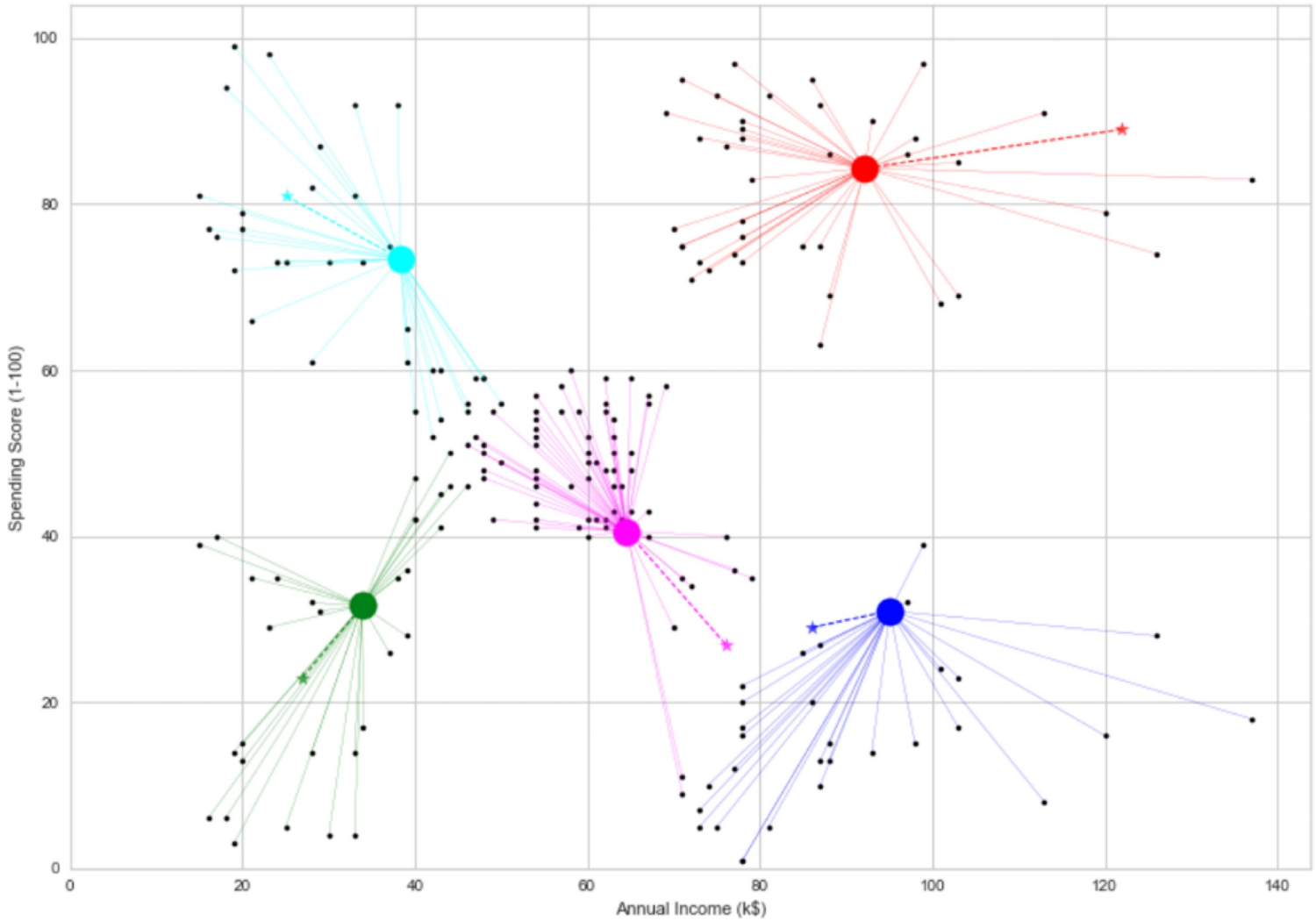


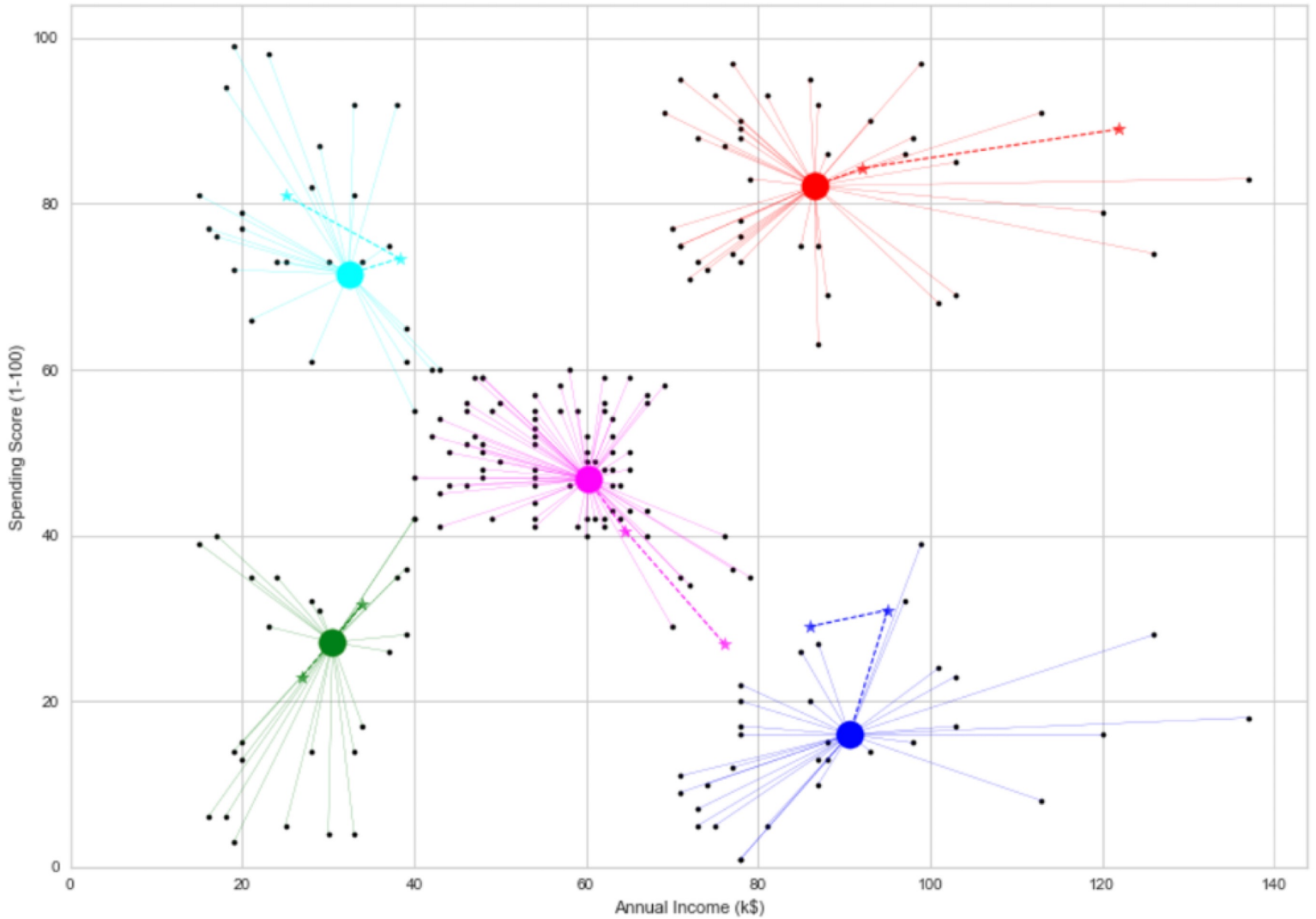
Illustration for K = 5



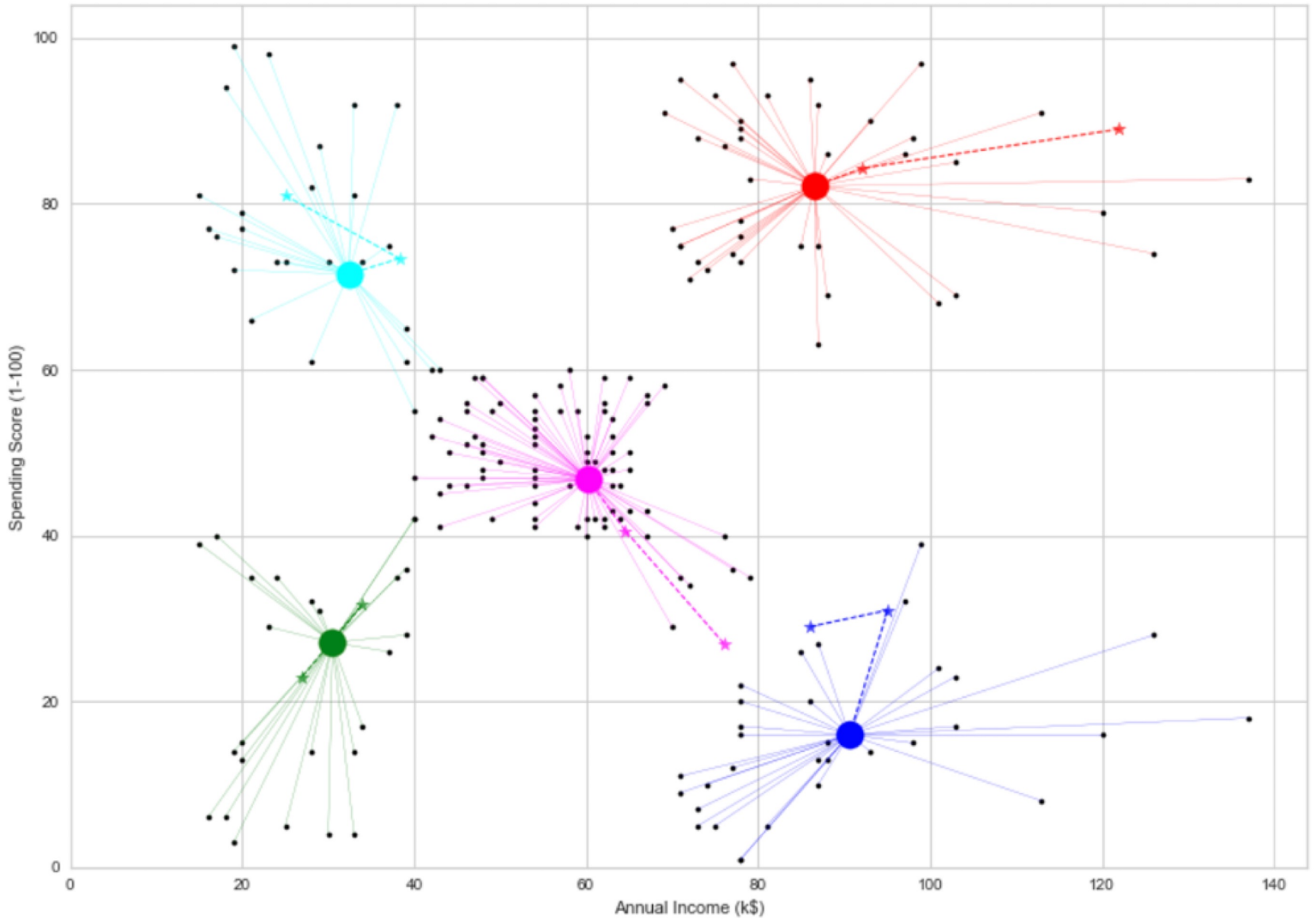
Iteration 1



Iteration 2

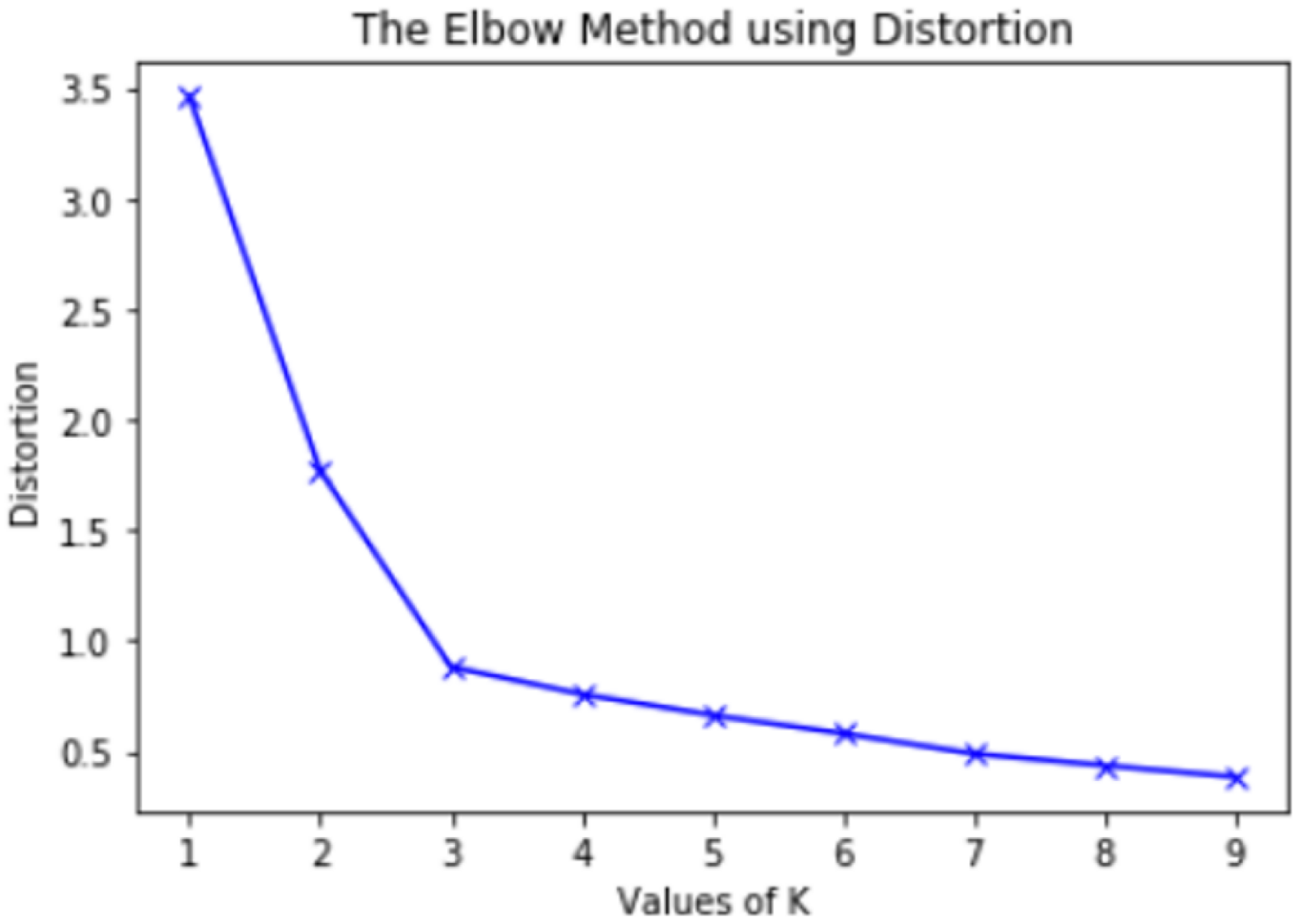


Iteration 3



Iteration 4 **Convergence attained**

The elbow method



K=4 is ideal here

Week 9 Recap: Topic Extraction using Clustering

- Step1: Pre-process all documents first
- Step2: Perform feature-extraction
- Step3: Cluster words using K-means
- Step4:
  - *Words at cluster centers and their neighbors represent topics*

For each cluster:

print words the cluster center

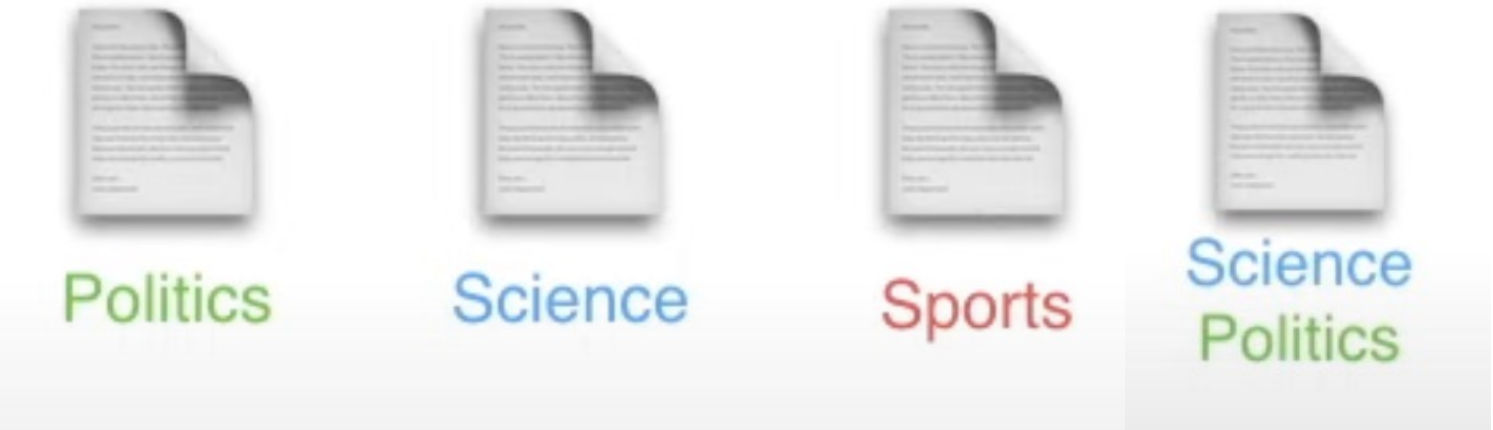
print N nearest neighbors of the cluster-center

Week 9 Recap: Topic Modeling

- Another unsupervised approach based on generative modeling
- Given a set of documents:

Week 9 Recap: Topic Modeling

- We are interested in learning:
  - Given a document
  - How topics are distributed in documents or **P(topic | document)**

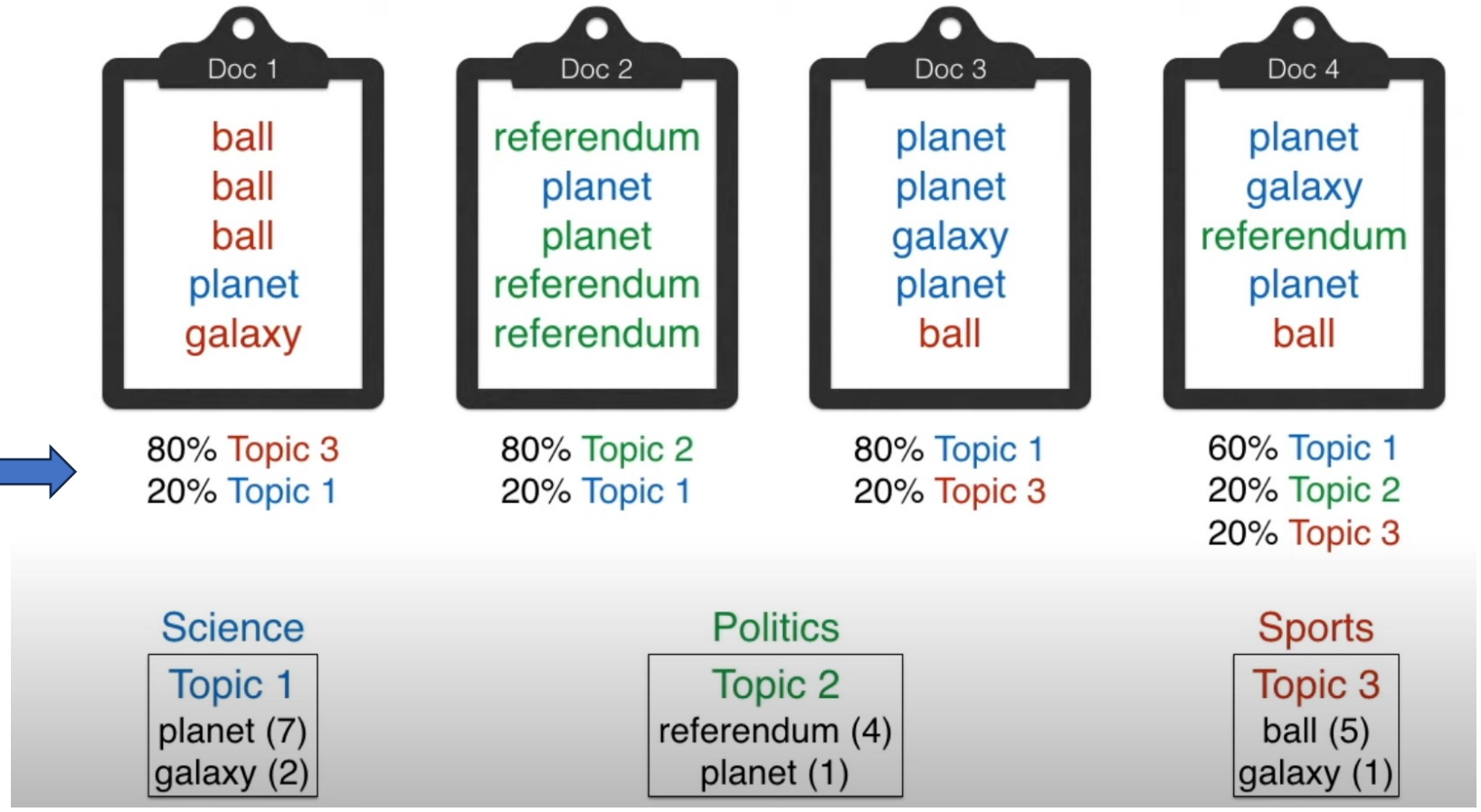


Week 9 Recap: Topic Modeling

- We are interested in learning:
  - Given a document
  - How topics are distributed in documents
  - And how each word contributes to topic **P(word | topic)**

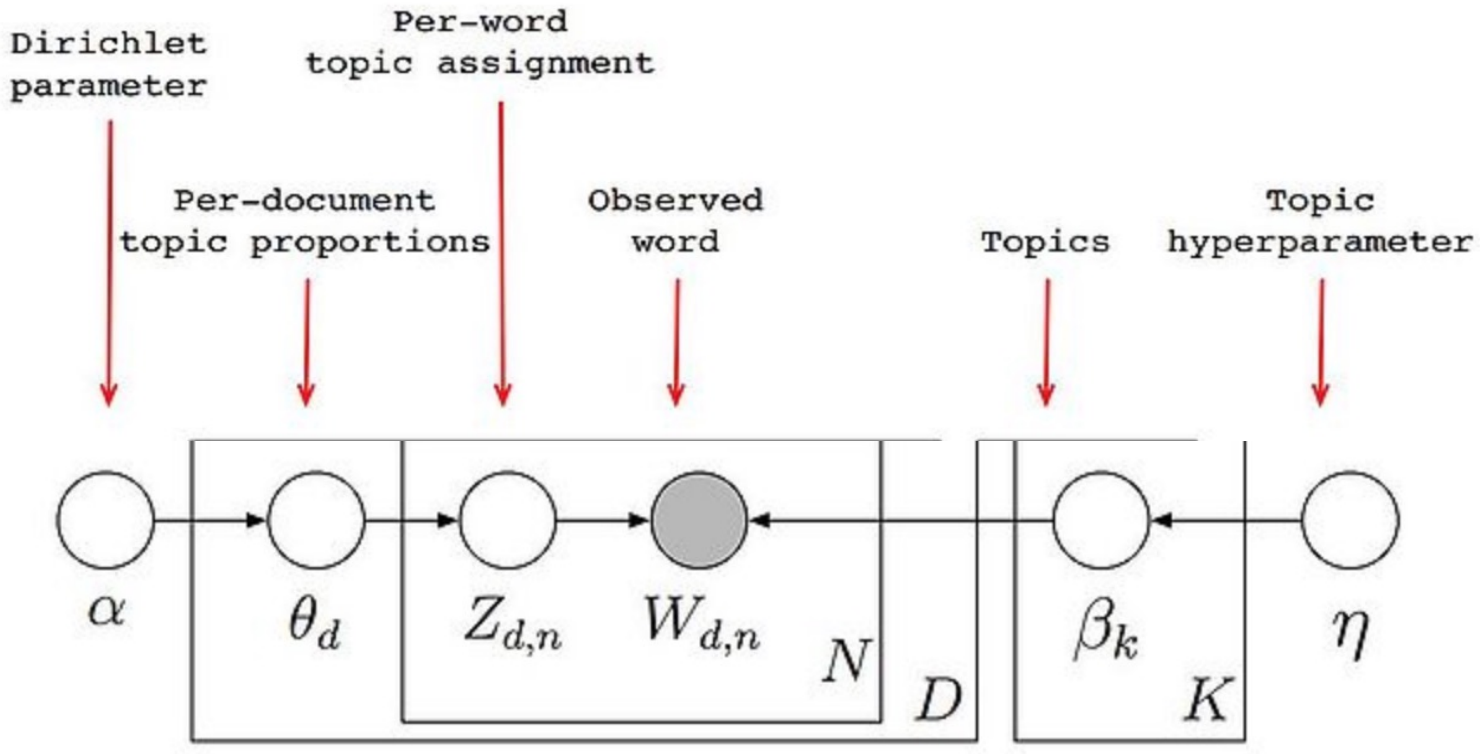
https://www.youtube.com/watch?v=BaW1uiCnjE

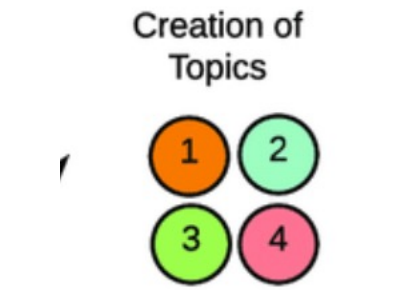
Overall

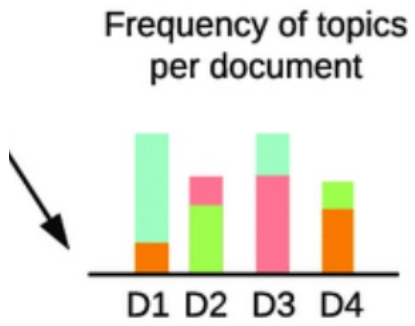

Doc 1
ball
ball
ball
planet
galaxy
Doc 2
referendum
planet
planet
referendum
referendum
Doc 3
planet
planet
galaxy
planet
ball
Doc 4
planet
galaxy
referendum
planet
ball
P(topic | document)
80% Topic 3
20% Topic 1
80% Topic 2
20% Topic 1
80% Topic 1
20% Topic 3
60% Topic 1
20% Topic 2
20% Topic 3
P(word | topic)
Science
Topic 1
planet (7)
galaxy (2)
Politics
Topic 2
referendum (4)
planet (1)
Sports
Topic 3
ball (5)
galaxy (1)

LDA: Estimating *P(word | topic), P(topic | document)*

Stands for **Latent Dirichlet Allocation**
It's a Bayesian Network



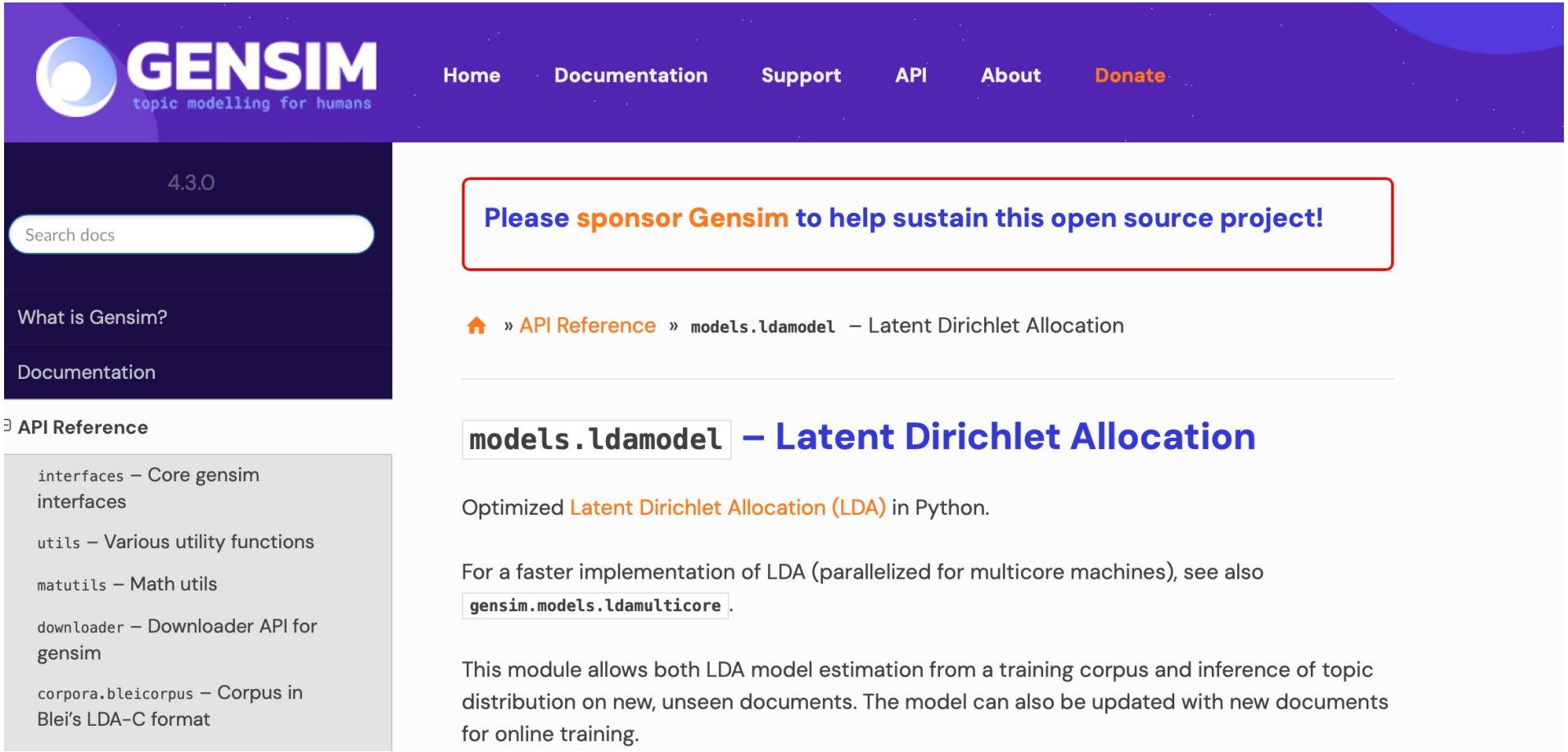




Evaluation of Topics

- Mostly Qualitative:
  - In unsupervised settings, we do not have labels like Sports, Politics ***etc***
  - Humans will have to assess the topic quality based on manual assessment
  - Sometimes, quantitative metrics such for topic coherence, perplexity etc are used.

In Python


GENSIM
topic modelling for humans
Home
Documentation
Support
API
About
Donate
4.3.0
Search docs
What is Gensim?
Documentation
API Reference
interfaces – Core gensim interfaces
utils – Various utility functions
matutils – Math utils
downloader – Downloader API for gensim
corpora.bleicorpus – Corpus in Blei's LDA-C format
Please sponsor Gensim to help sustain this open source project!
» API Reference » models.ldamodel – Latent Dirichlet Allocation
models.ldamodel – Latent Dirichlet Allocation
Optimized Latent Dirichlet Allocation (LDA) in Python.
For a faster implementation of LDA (parallelized for multicore machines), see also gensim.models.ldamulticore.
This module allows both LDA model estimation from a training corpus and inference of topic distribution on new, unseen documents. The model can also be updated with new documents for online training.

So far in I320D – Text Mining and NLP

- W1. Language and Ambiguity
- W2. Basics of Text Data and Linguistic Concepts
- W3. Text Preprocessing Techniques
- W4. Lexical Analysis
- W5. Syntax Analysis
- W6. Information Extraction

W7. Machine Learning Methods for NLP
W8. Unsupervised ML and Topic Modeling Basics
W10-W11. Deep learning for NLP
W12. NLP Applications
W13. Small and Large Language Models and Prompt Engineering Basics
W14. Knowledge Networks
W15. Evaluation Metrics

This Week's Topics

- Introduction to Deep Learning for NLP
- Feed Forward, Recurrent Networks and Transformers

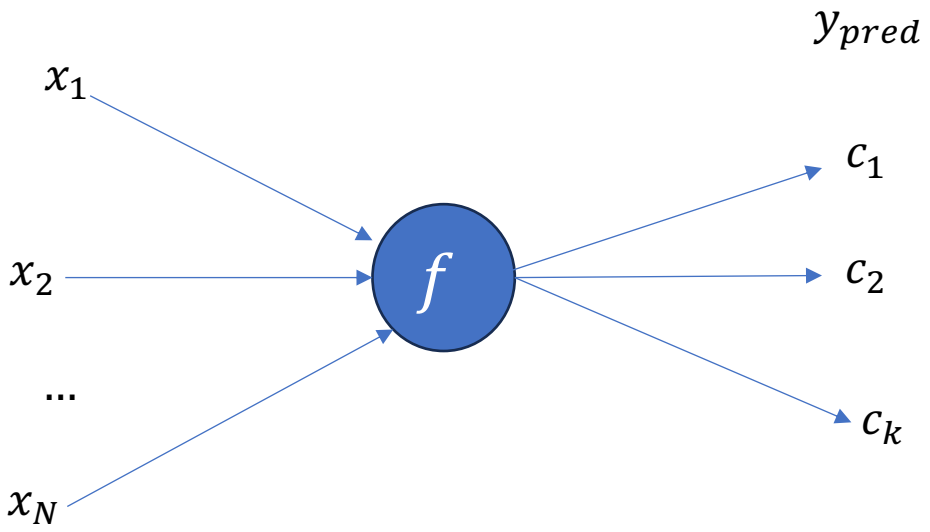
Introduction to Deep Learning

- Deep Learning:
  - Multi layered interconnected neural networks
  - Great for supervised learning

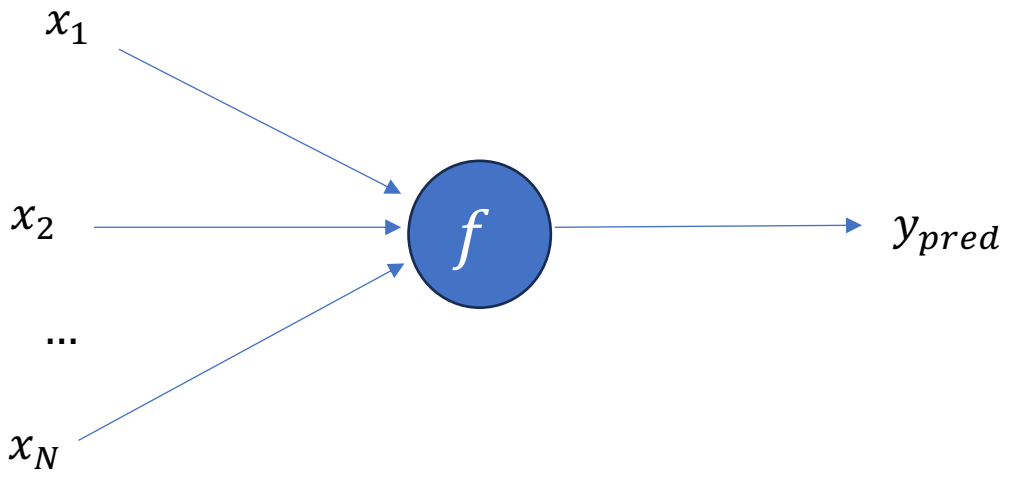
Supervised Learning : Recap

- Learning from “Labeled training data”
- Labeled data = bunch of examples {Input, Expected Output}
- Classification or Regression depending on the outcome type
  - **Classification:** When the expected output type is “**categorical**”
  - **Regression:** When the expected output type is numeric (real number)

Supervised Learning



K-class classification



Regression

Training objective: to learn decision function $f(.)$ from data

What is Training?

- **Machine Learning:** automatically learning programs (i.e.,decision function) from experiences (data)

- **Training an ML model:** Define a decision function that **minimizes the error** on the training dataset

What is training?

- For a set of M training examples, each containing N features , minimize the average error on examples

$$\underset{f}{\text{minimize}} \quad \frac{1}{M}\sum_{\text{i}=1}^{\text{M}} \text{Err}(\text{y}^{\text{i}}_{\text{actual}}, \text{y}^{\text{i}}_{\text{predicted}})$$

$$\Rightarrow \quad \underset{f}{\text{minimize}} \quad \frac{1}{M}\sum_{\text{i}=1}^{\text{M}} \text{Err}(\text{y}^{\text{i}}_{\text{actual}}, \text{f}(x_1^i, x_2^i, \ldots, x_N^i))$$

What is Err( )?

- Mathematical measure that quantifies how well a machine learning model's predictions match the actual (true) values of the data it's trying to learn from.
- Often referred to as "**Loss**" function or "**Empirical Risk**" in ML
- Many possibilities
- Let's start with a simple (and elegant) error function

$$(y_{\text{predicted}} - y_{\text{actual}})^2$$

Regression: What is training?

- For a set of M training examples, each containing N features

$$\underset{f}{\text{minimize}} \; \frac{1}{M} \sum_{\text{i}=1}^{\text{M}} (\text{y}_{\text{actual}}^{\text{i}} - \text{y}_{\text{predicted}}^{\text{i}})^2$$

Mean squared error

Parametric Function

$$\Rightarrow \; \underset{f}{\text{minimize}} \; \frac{1}{M} \sum_{\text{i}=1}^{\text{M}} (\text{y}_{\text{actual}}^{\text{i}} - \text{f}(x_1^i, x_2^i, \ldots, x_N^i))^2$$

Error for Classification

- **Cross Entropy Error**

$$Err = -\sum_{c \,\epsilon\, C} y^{c}_{actual} . log\ y^{c}_{predicted}$$

Where C is a collection of all possible classes

What is training – classification?

- For a set of M training examples, each containing N features

$$\underset{f}{\text{minimize}} \quad -\frac{1}{M}\sum_{i=1}^{M}\sum_{c \in C} y_{actual}^{i,c} \cdot log\, y_{predicted}^{i,c}$$

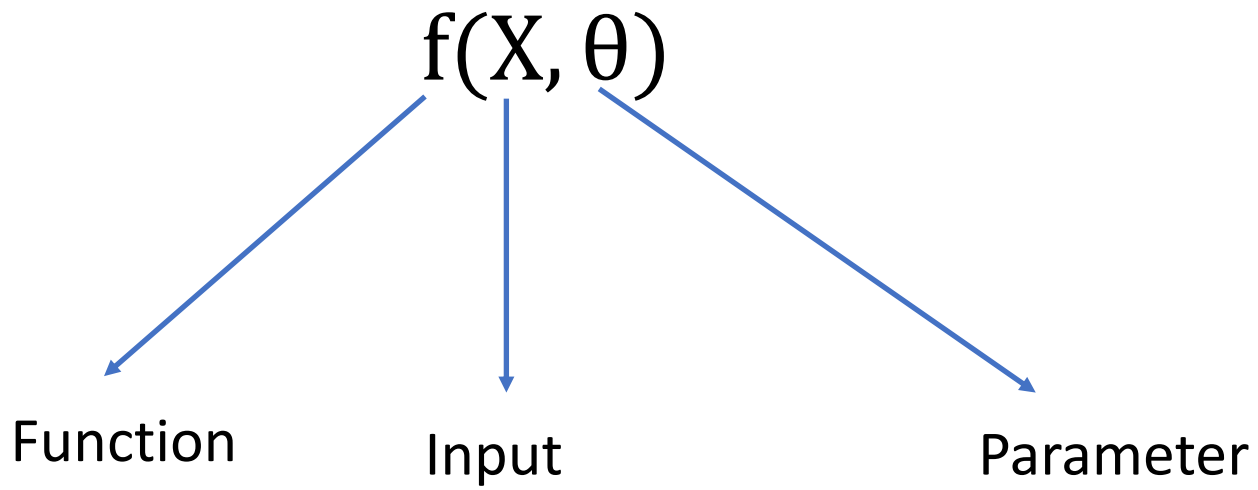
$$= \underset{f}{\text{minimize}} \quad -\frac{1}{M}\sum_{i=1}^{M}\sum_{c \in C} y_{actual}^{i,c} \cdot \log(\mathrm{f}^{\mathrm{c}}(x_1^i, x_2^i, \ldots, x_N^i))$$

What is f( )

- Can be any mathematical function
- BUT we restrict it to a certain class of functions
- Example:
  - Naïve Bayes: Models based on Bayes' theorem
- Another class of f():
  - Perceptron – A Parametric decision function

Parametric functions – a broader class of f( )

- Function characterized by a set of parameters
- Often written as



Parametric functions

- Example

$$f(X, \theta) = \log_a(x)$$

Parameter(s) = θ = a

Parametric functions – Exercise

- Example

$$f(X, \theta) = \exp(a, x)$$

Parameter(s) = θ ?

Parametric functions – Exercise

- Example

$$f(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Parameter(s) = θ ?

Parametric functions – Exercise

- Example

$$f(X, \theta) = w_1 x_1 + w_2 x_2 + \cdots + w_N x_N + \beta$$

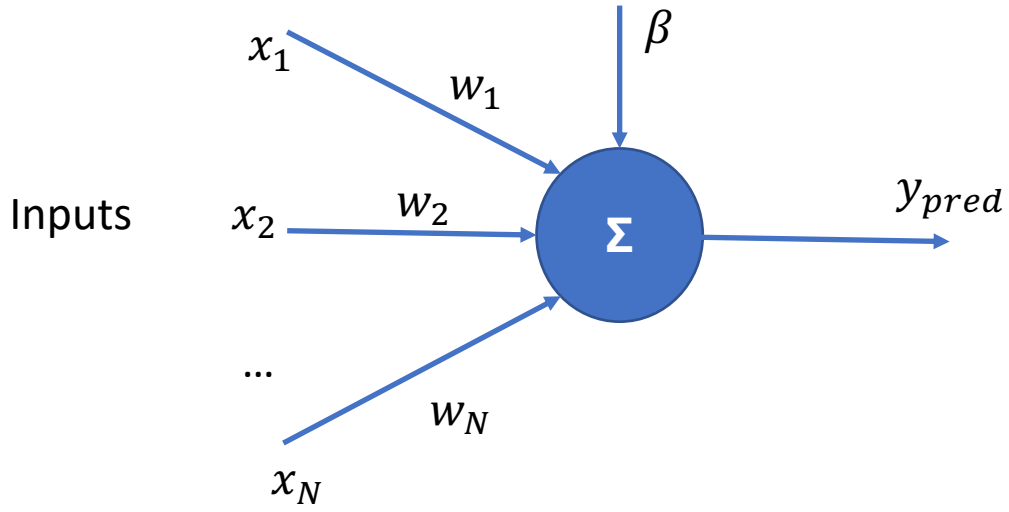
Input(s) = ?

Parameter(s) = ?

Introduction to Perceptrons

Perceptrons

- Simple parameterized decision functions that can act as building blocks for complex decision functions



$$\mathbf{y_{pred} = w_1 x_1 + w_2 x_2 + \cdots + w_N x_N + \beta}$$

OR

$$\mathbf{y = W^T X + \beta}$$

$$\Theta = [w_1, w_2, \ldots, w_N]$$

Training a perceptron

Training with linear parametric decision functions

$$\underset{f}{\text{minimize}} \sum_{\text{i=1}}^{\text{M}} (\text{y}_{\text{actual}}^{\text{i}} - \text{f}(x_1^i, x_2^i, \ldots, x_N^i))^2$$

- **In this case**

$$\underset{w_1, w_2, \beta}{\text{argmin}} \sum_{\text{i=1}}^{\text{M}} (\text{y}_{\text{actual}}^{\text{i}} - (\text{w}_1 x_1 + w_2 x_2 + \beta))^2$$

How to get the right set of Ws?

- **Randomly guess Ws?**

Problem with random guessing

- Doesn’t guarantee convergence
- So many values to guess and for each value of ***w,*** so many possibilities
- How about findings Ws with finite number of trials:
  - **Gradient Descent algorithm**

Gradient Descent Algorithm

1. Initialize Ws with random values
2. Predict $y_{pred} = w_1.x_1 + w_2.x_2 + \cdots + w_N.x_N + \beta$ for each example in training data
3. Compute Mean Squared Error (Err) on all training examples
4. Compute the gradient of Error, say $\nabla(Err)$ with respect to all Ws
5. Update the weights

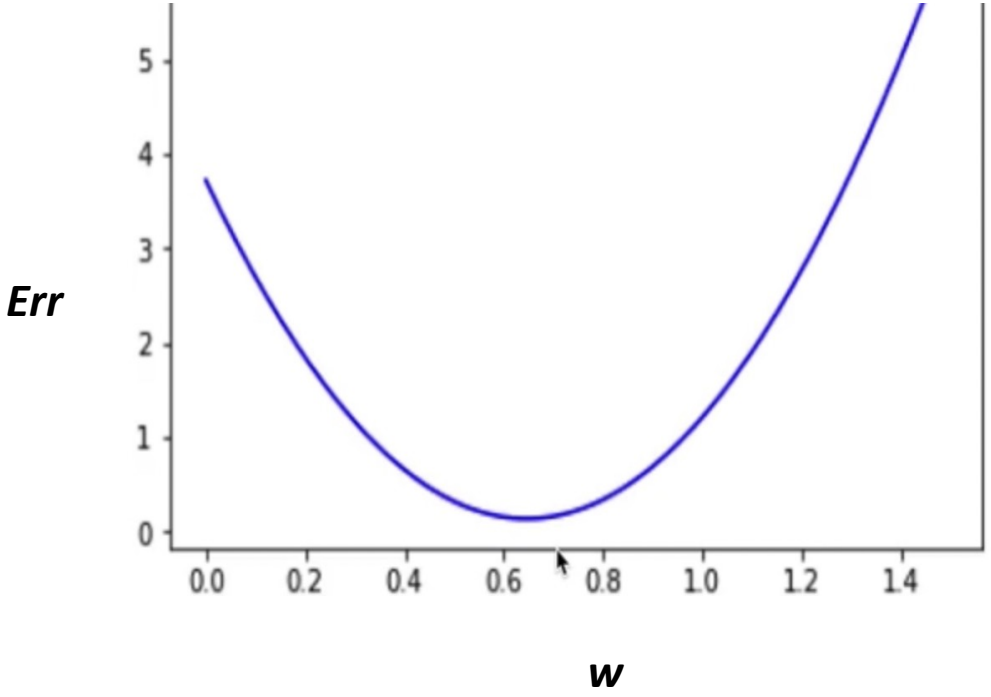
$\boldsymbol{\eta} \rightarrow$ *A user defined paramter (a.k. a Hyperparameter) also called as "learning rate"*

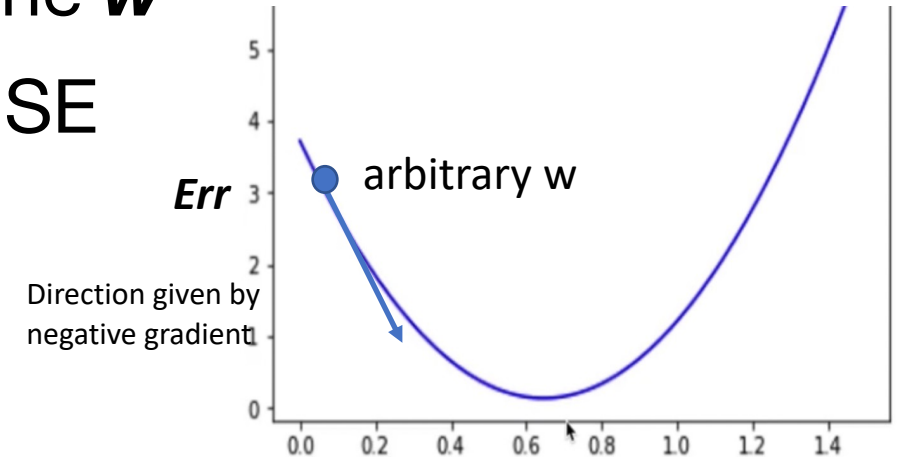
$$W^{new} = W^{old} - \eta\nabla(Err)$$

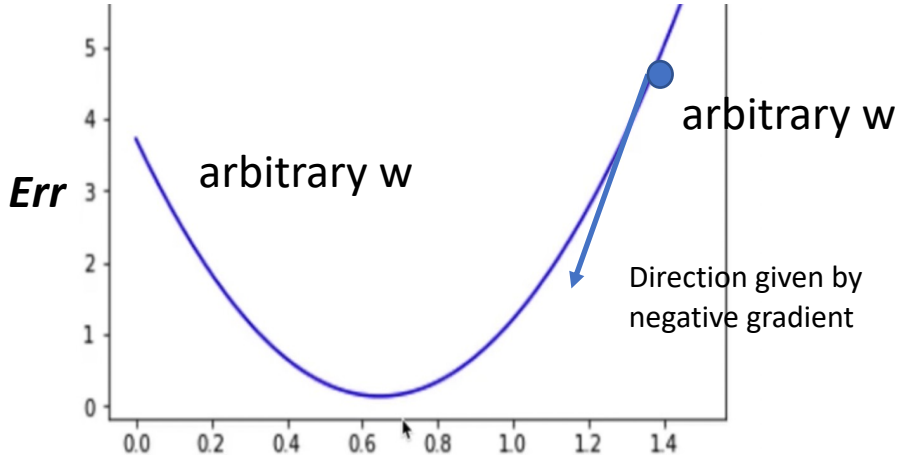
6. Repeat steps 2-5 with W_{new} until convergence (i.e.,$W^{new} \sim W^{old}$)

Gradient Descent: Intuition

- Consider simple function with one ***w***
- Assume the following plot for MSE

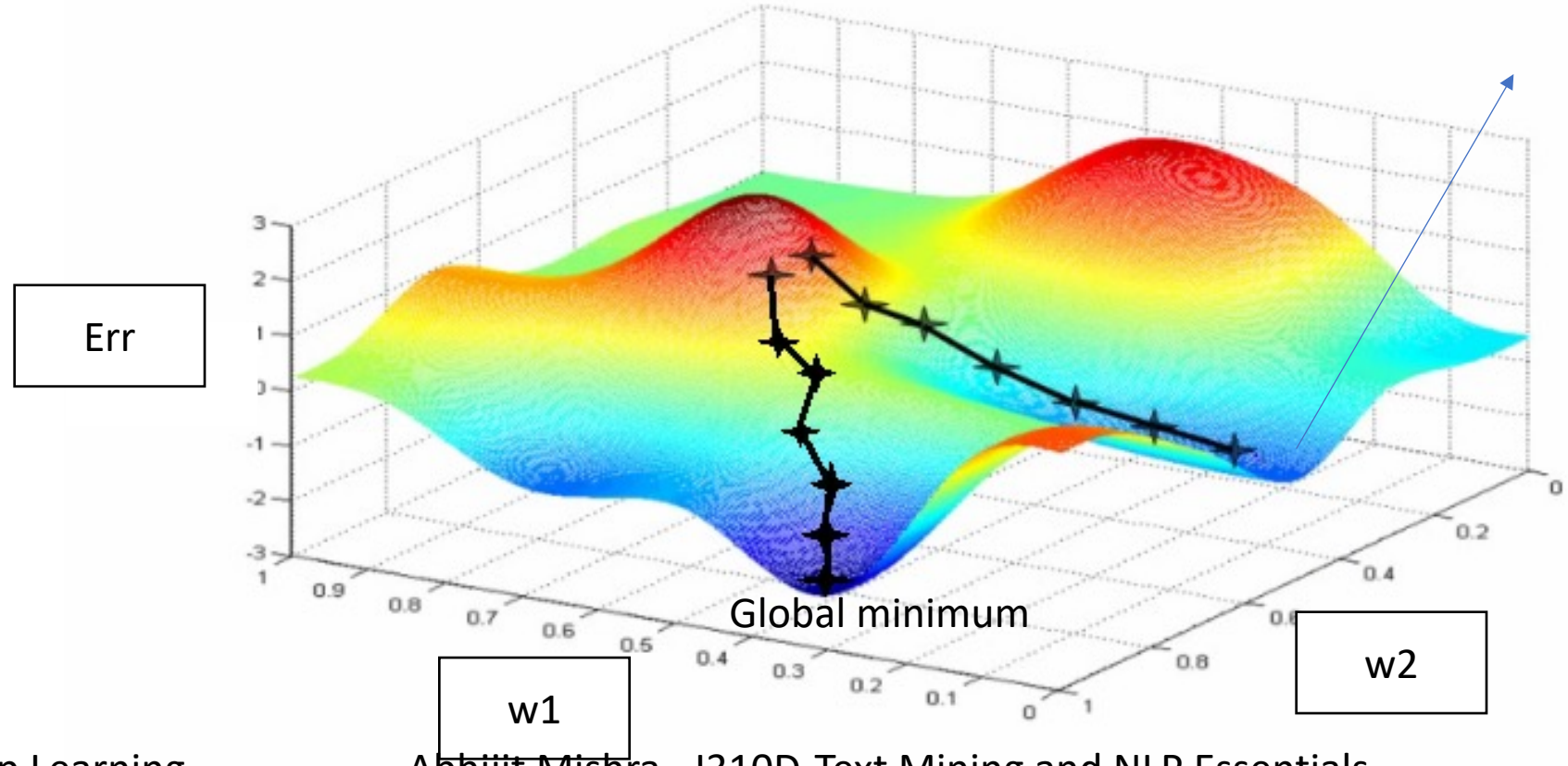






Gradient Descent: Intuition (…)

- Consider another simple function with two parameters w1, w2
- Assume the following 3D plot for MSE



Visualizer

- https://uclaacm.github.io/gradient-descent-visualiser/#playground

Some Pointers

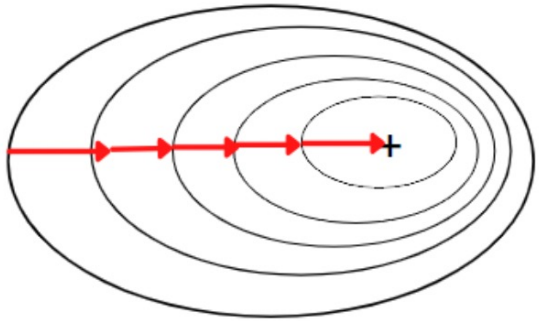
“Where you start and how far you go in every step matters”

- Better initialization and learning rates yield faster convergence and better minima
- Decided by trial and error (or through model validation)

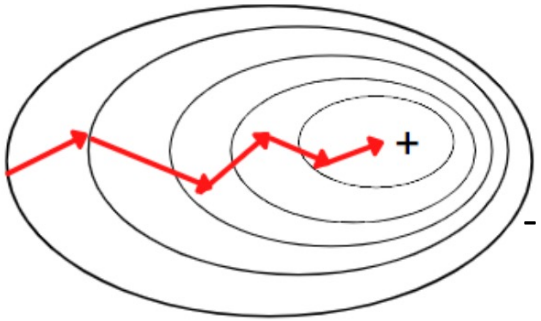
Gradient Descent Types

- Whole training data passed at once
- MSE on complete training dataset

Batch Gradient Descent

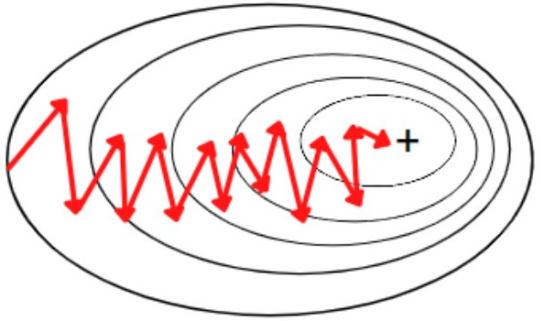


Mini-Batch Gradient Descent



- Training dataset divided into several segments (mini-batches)
- One mini-batch picked at time to compute MSE and used for gradient computation

Stochastic Gradient Descent



- Only one example is used at a time for computing MSE

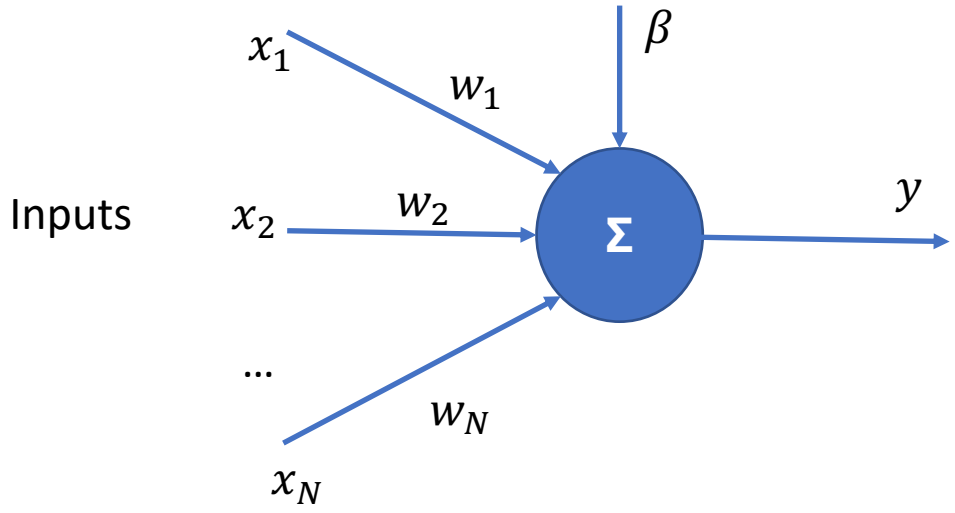
https://www.analyticsvidhya.com/blog/2022/07/gradient-descent-and-its-types/

Questions?

Perceptron Activation Functions

Back to Perceptrons

- Simple decision functions that can act as building blocks for complex decision functions



$$y = w_1x_1 + w_2x_2 + \cdots + w_Nx_N + \beta$$

OR

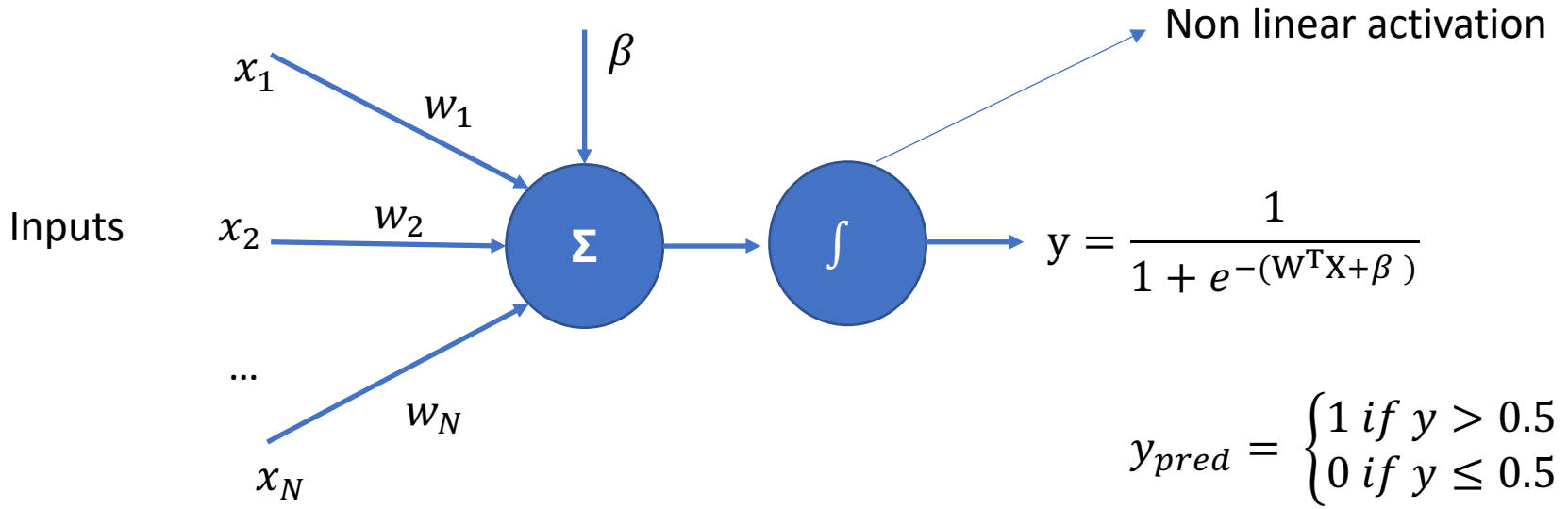
$$y = W^TX + \beta$$

Which model is this? (Recall from previous lectures)

Linear Regression

Can we make the perceptrons non-linear?

- Simple decision functions that can act as building blocks for complex decision functions

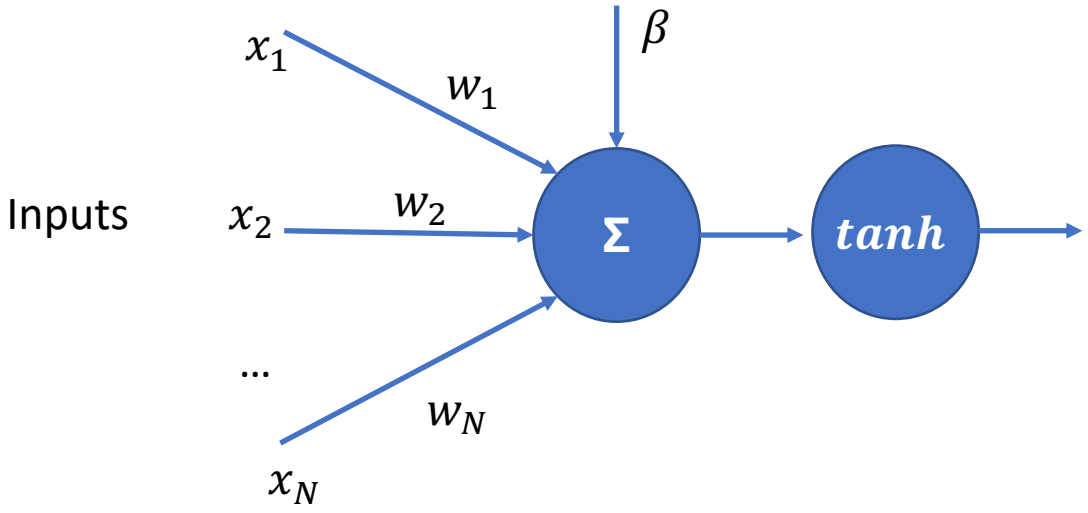


Which model is this? (Recall from Machine Learning lectures) – classification or regression?

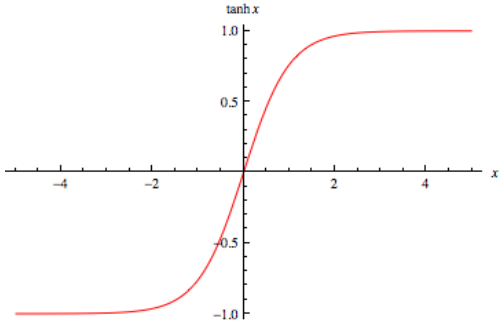
Logistic Regression (classification)

Perceptrons

- Simple decision functions that can act as building blocks for complex decision functions



$$y = \frac{e^{2(W^T X+\beta)} - 1}{e^{2(W^T X+\beta)} + 1}$$



tanh

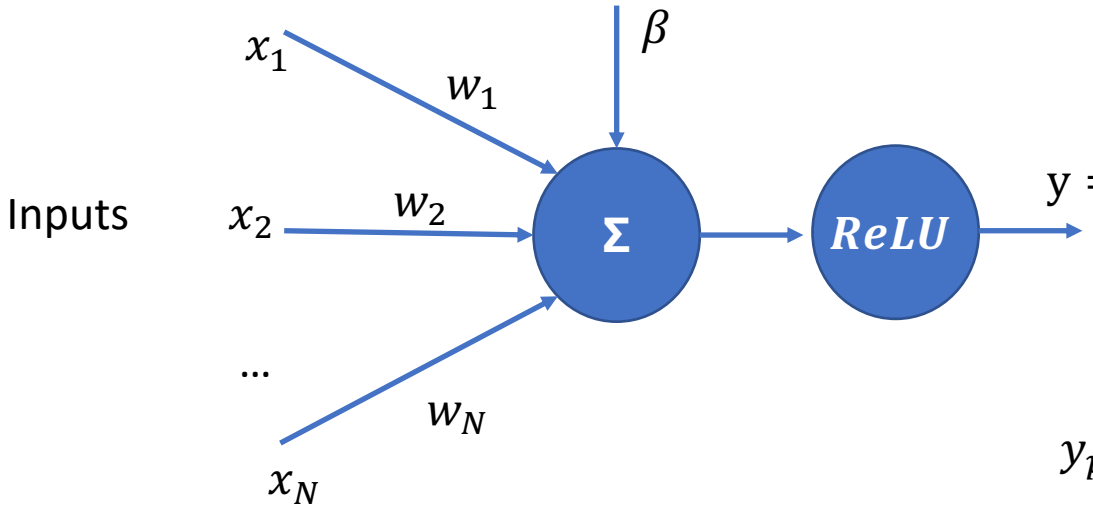
$$y_{pred} = \begin{cases} 1 \; if \; y > 0 \\ -1 \; if \; y \leq 0 \end{cases}$$

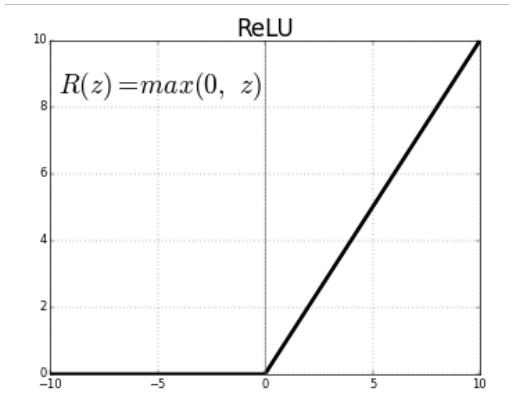
Which model is this? classification or regression?

Classification

Perceptrons

- Simple decision functions that can act as building blocks for complex decision functions





$$y_{pred} = \begin{cases} W^T X + \beta \text{ if } \mathrm{W^T X} + \beta > 0 \\ 0 \text{ otherwise} \end{cases}$$

Which model is this? classification or regression?

Classification + Regression ?

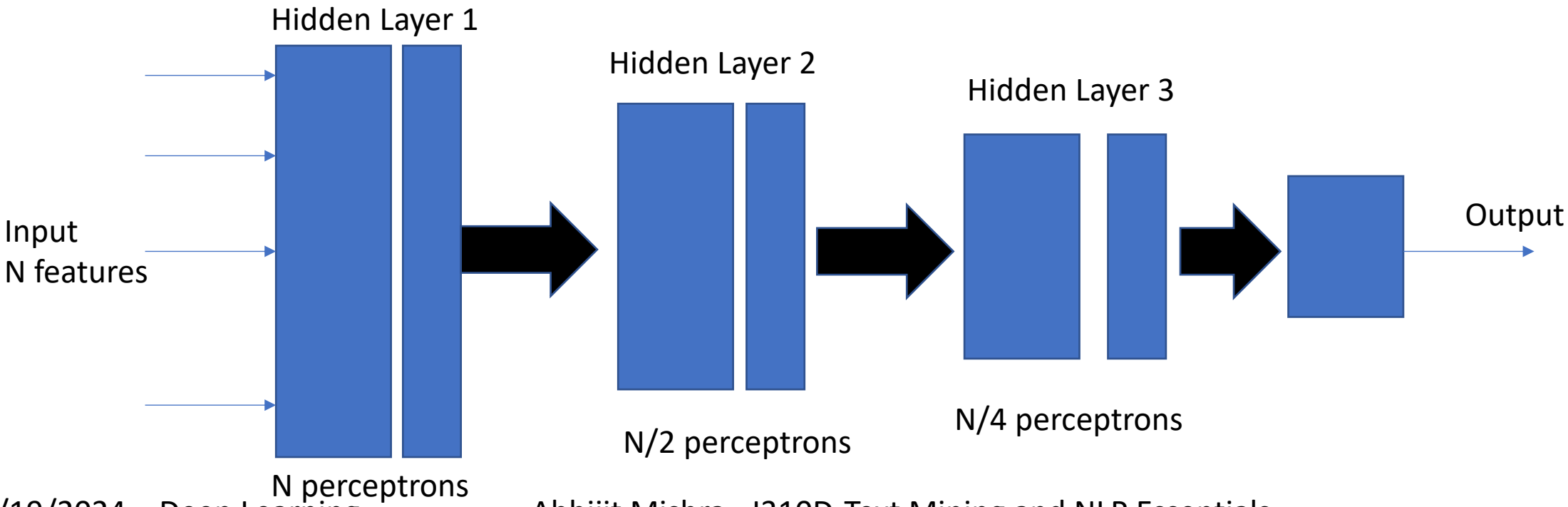
Relation with Biological Neurons

Dendrons and axons (Memory and Activations / Information Filtering)

Neural Networks

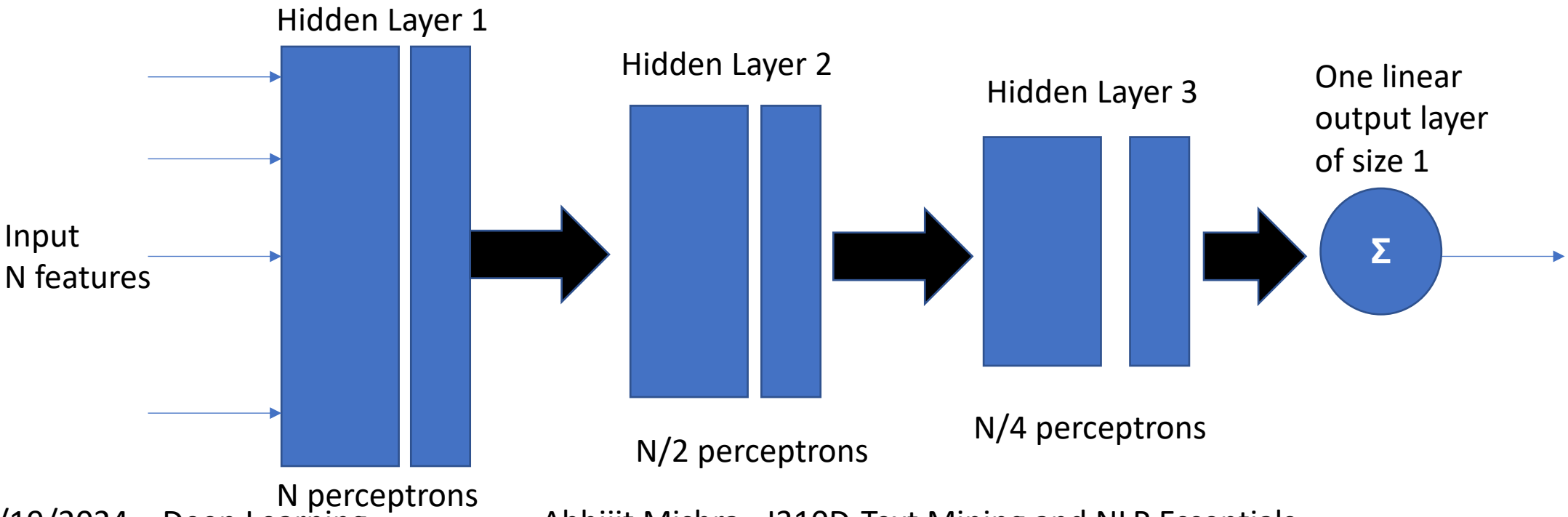
Feed forward neural network

- Layers of perceptrons / artificial neurons
- Output of every perceptron is fed to the next layer



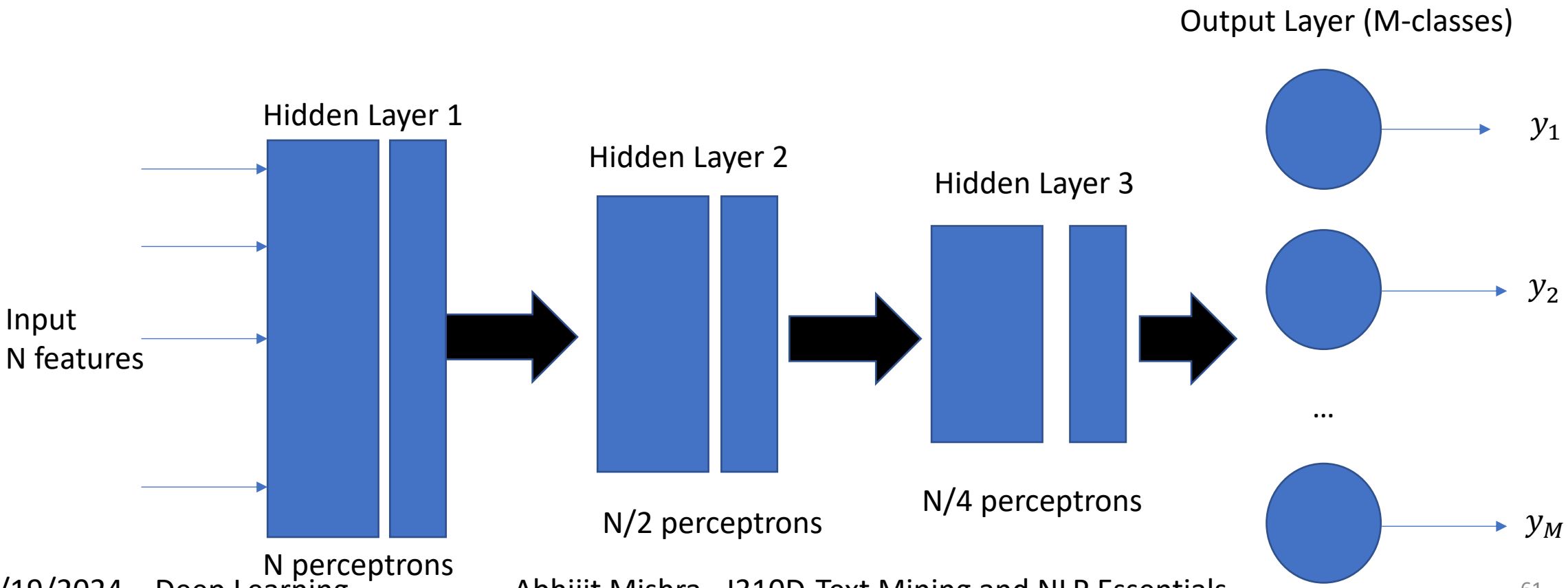
Feed forward network

- Regression



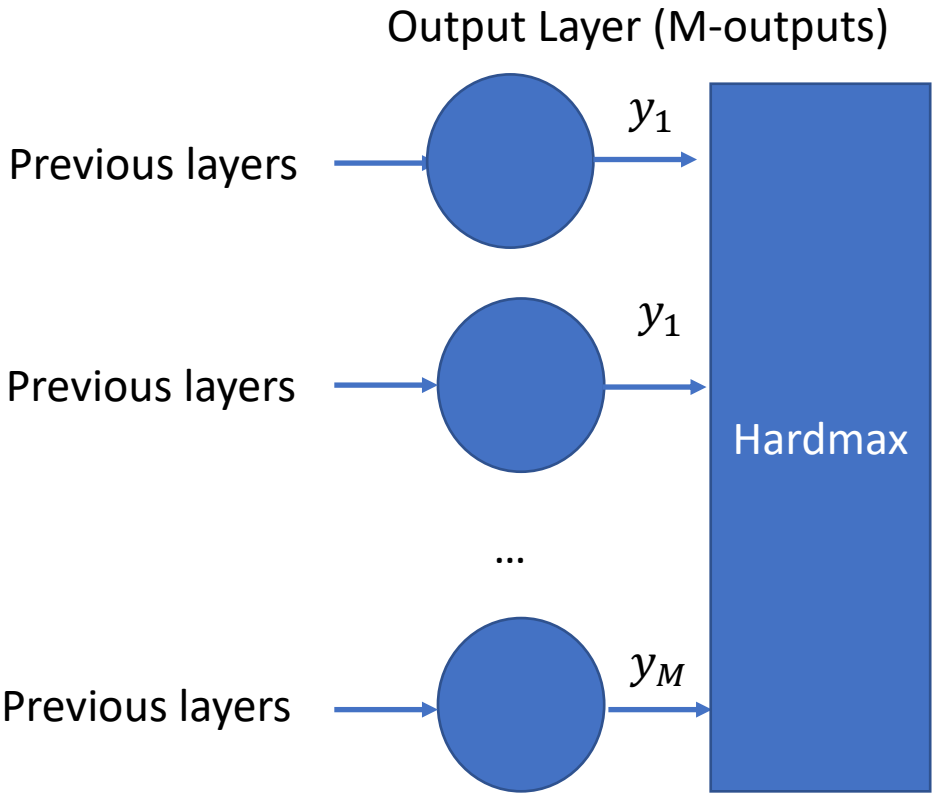
Feed forward network

- M-class classification



How to handle multiclass inputs?

- For multiclass classification with M classes– scale using HardMax



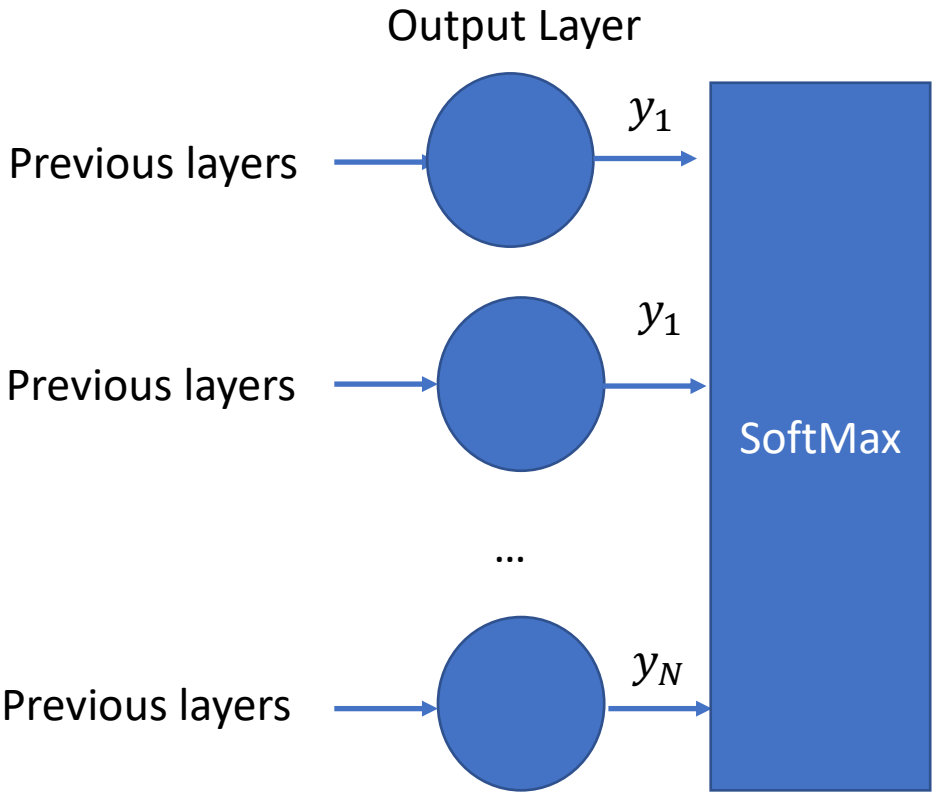
$$y_1 = \frac{y_1}{y_1 + y_2 + y_3 + \cdots y_M}$$

$$y_2 = \frac{y_2}{y_1 + y_2 + y_3 + \cdots y_M}$$

$$y_M = \frac{y_M}{y_1 + y_2 + y_3 + \cdots y_M}$$

How to handle multiclass inputs?

- Or an even smoother scaling function– SoftMax



$$y_1 = \frac{e^{y_1}}{e^{y_1} + e^{y_2} + e^{y_3} \ldots + e^{y_M}}$$

$$y_2 = \frac{e^{y_2}}{e^{y_1} + e^{y_2} + e^{y_3} \ldots + e^{y_M}}$$

$$y_M = \frac{e^{y_M}}{e^{y_1} + e^{y_2} + e^{y_3} \ldots + e^{y_M}}$$

In general…

- The final function learned by a network of perceptrons is a stacked version of individual functions represented by each perceptron

$$y = h(g(f(x1, x2, x3, \ldots x_N)))$$

where $f(.), g(.), h(.)$ are individual perceptrons

In general…

A multi-layer perceptron can be used to approximate any function.

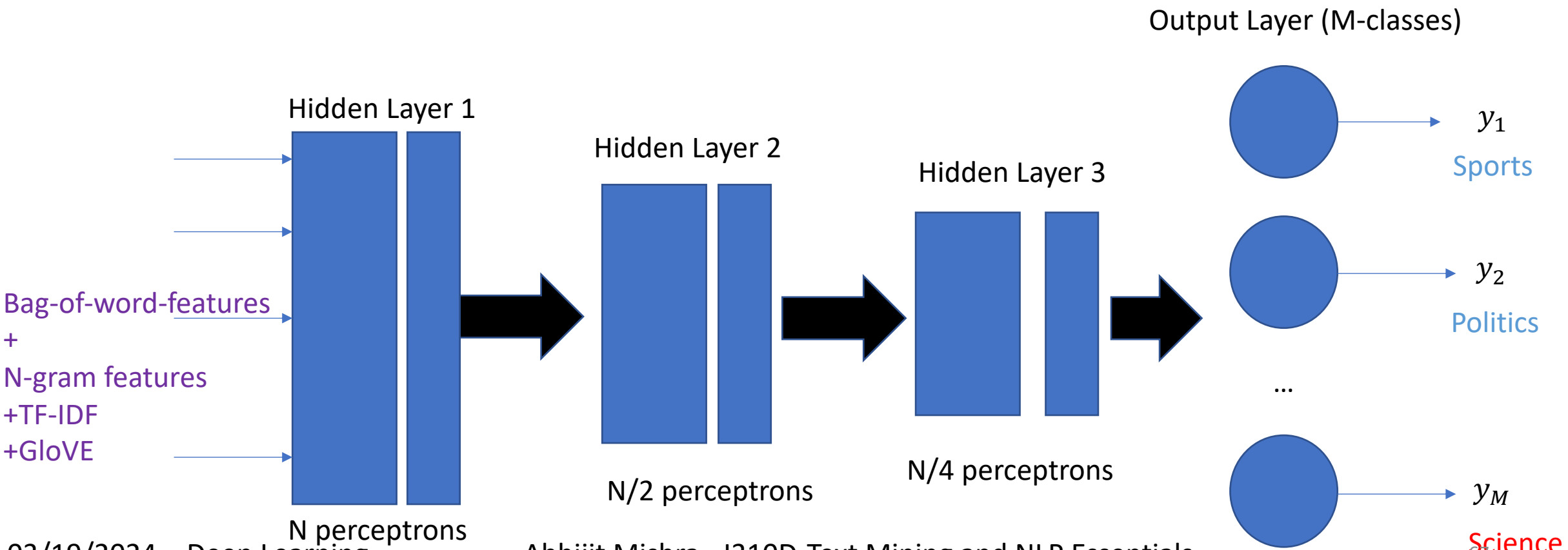
Universal approximation theorem

And again

- We introduce non-linearity in TWO ways:
  - By using activation functions that are non-linear
  - By stacking networks in a layered manner

Feed Forward Network: Text Classification Example

- M-class classification



Why Neural Networks

OR Why “yet another classification model”?

Three learnings from traditional ML

1. Feature Engineering is “hard”

- Which columns to select from tables?
- What to do when input is unstructured (e.g., images/text)

Loves the German bakeries in Sydney. Together with my imported honey it feels like home	Positive
@VivaLaLauren Mine is broken too! I miss my sidekick	Negative
Finished fixing my twitter...I had to unfollow and follow everyone again	Negative
@DinahLady I too, liked the movie! I want to buy the DVD when it comes out	Positive
@frugaldougal So sad to hear about @OscarTheCat	Negative
@Mofette briliant! May the fourth be with you #starwarsday #starwars	Positive
Good morning thespians a bright and sunny day in UK, Spring at last	Positive
@DowneyisDOWNEY Me neither! My laptop's new, has dvd burning/ripping software but I just can't copy the files somehow!	Negative

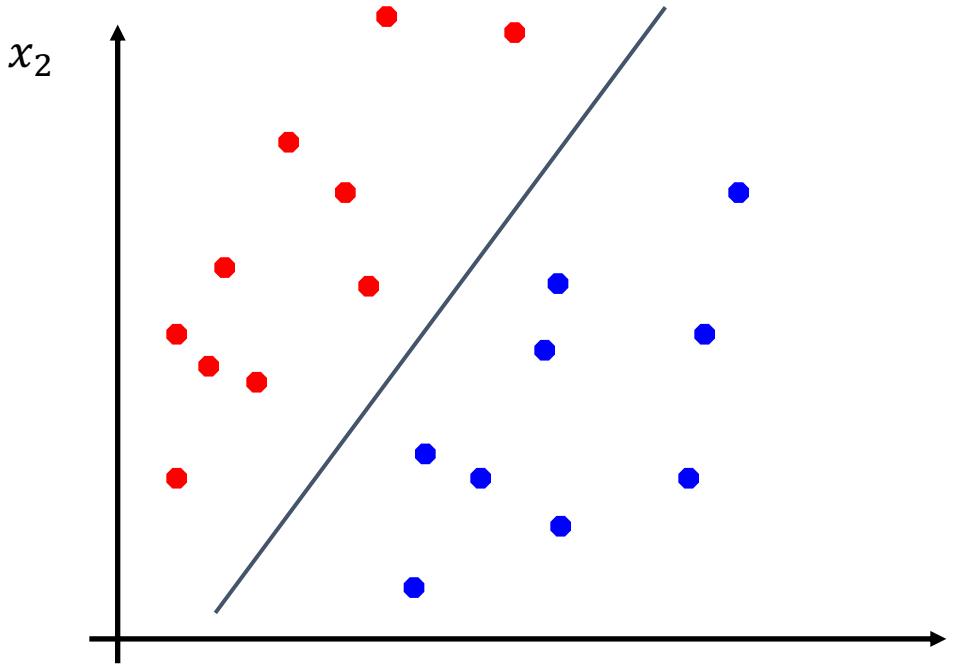
Example: Text classification

How to extract meaningful feature vectors from the input text?

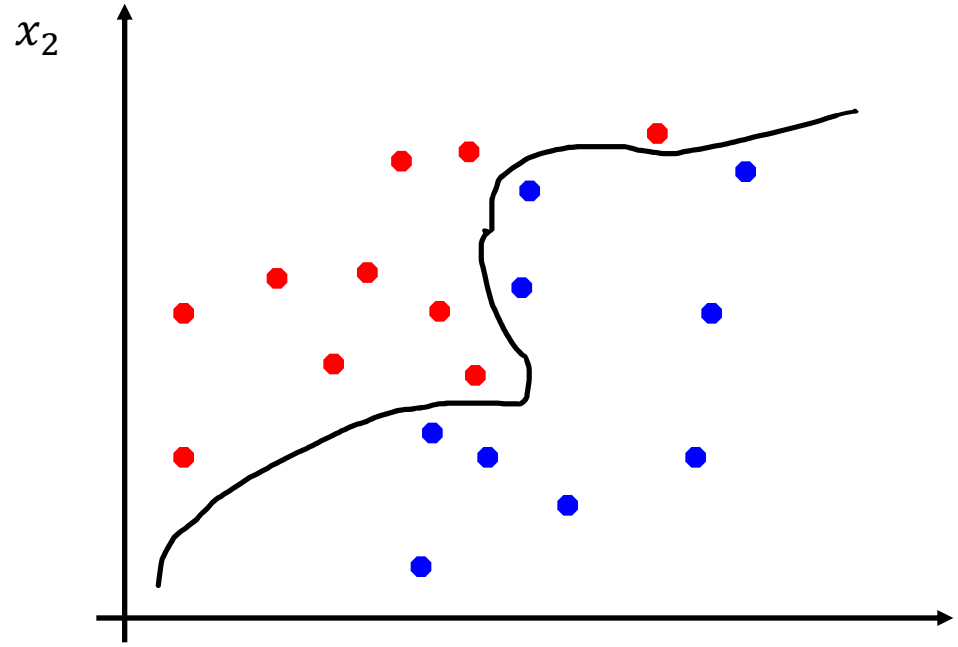
Three learnings from traditional ML

2. Linear functions do not fit well on most data

- Data is often non-linearly separated (especially unstructured data)



What we expect



What we often get

Three learnings from traditional ML

3. Complex input and output spaces are hard to handle
   - **Complex Inputs ->** Huge Feature Vectors -> Overfitting
   - **Complex Outputs:** Multiclass classification
     - Not truly done in One-One or One-Rest methods
     - We are merely extending binary classifiers

Potential solution?

Stack many "basic" classifiers and regressors together and connect them to form a network?

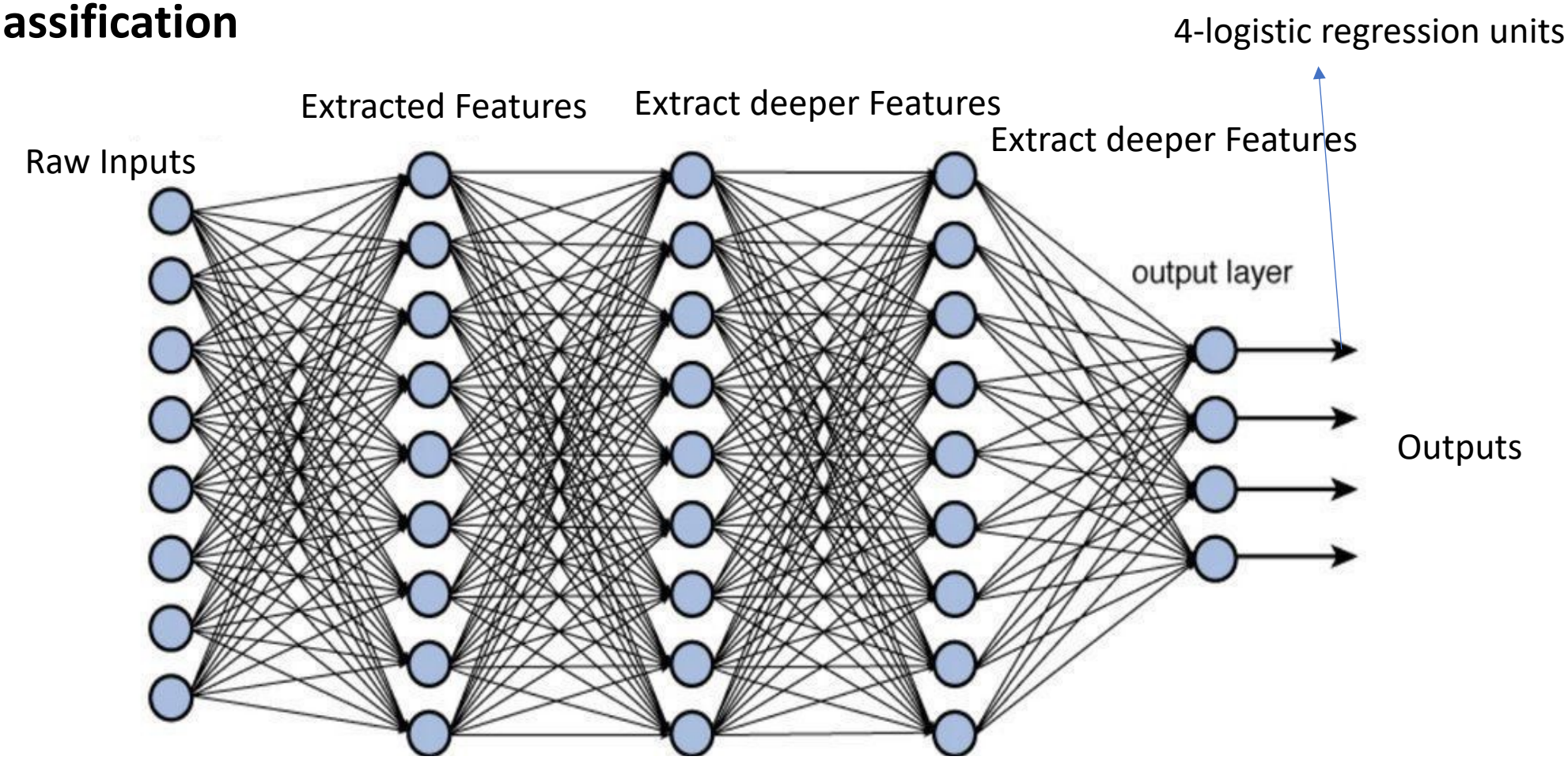
In other words – make the network DEEP

The function representing the network will be complex enough:

- To tackle any form of non-linearity
- Can yield multiple outputs
- Can learn to automatically extract meaningful features from raw inputs (say pixels or words)

Something like this

4-class classification



Training Data: <raw_input, output>

Training a Neural Network

Training Neural Networks: The Back Propagation Algorithm

Training one Perceptron – Gradient Descent

1. Initialize Ws = $[w_1^{old}, w_2^{old}, \ldots, w_N^{old}]$ with random values
2. Predict $y_{predicted} = f(X)$ for each example using the perceptron
3. Compute Mean Squared Error (Err) on all training examples
4. Compute the gradient of Error, say $\nabla(Err)$ with respect to all Ws
5. Update the weights

$$w_1^{new} = w_1^{old} - \eta \frac{\partial(Err)}{\partial w_1}, \eta > 0, \text{a. k. a learning rate}$$

$$w_2^{new} = w_2^{old} - \eta \frac{\partial(Err)}{\partial w_2},$$

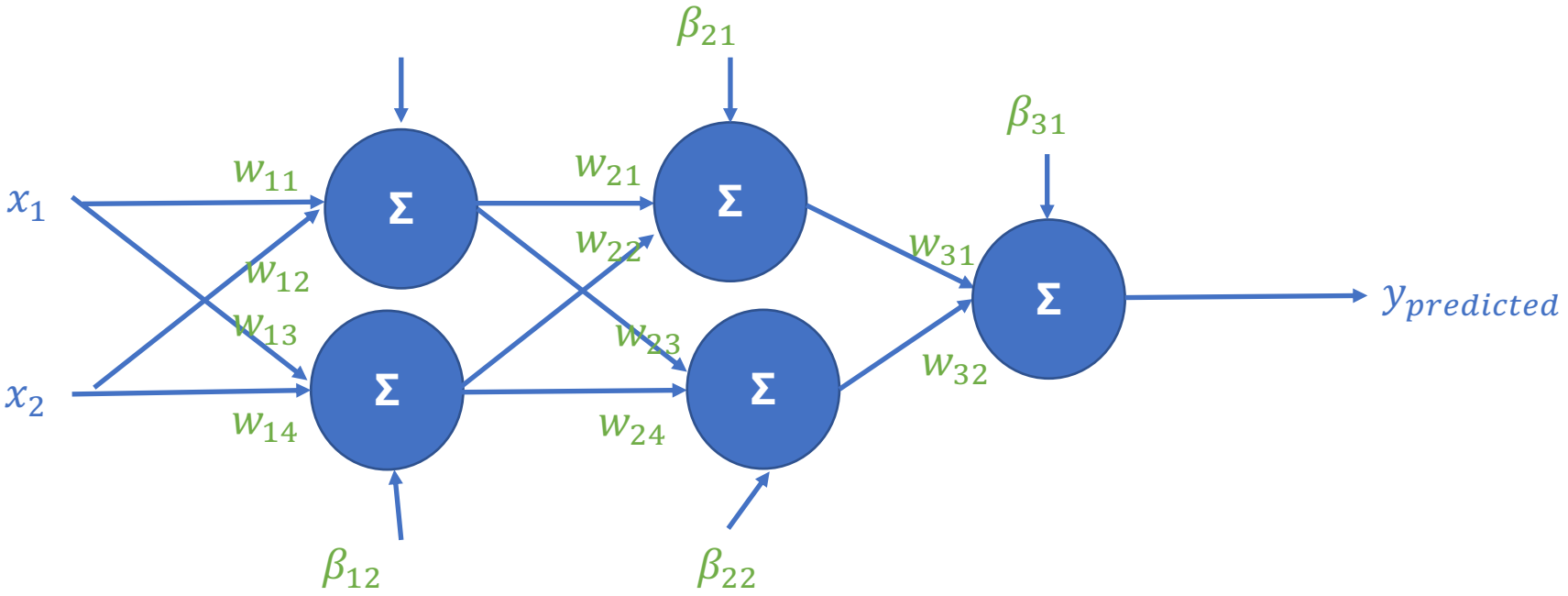
$$\ldots$$

$$w_N^{new} = w_N^{old} - \eta \frac{\partial(Err)}{\partial w_N}$$

5. Repeat steps 2-5 with W_{new} until convergence (i.e.,$W^{new} \sim W^{old}$)

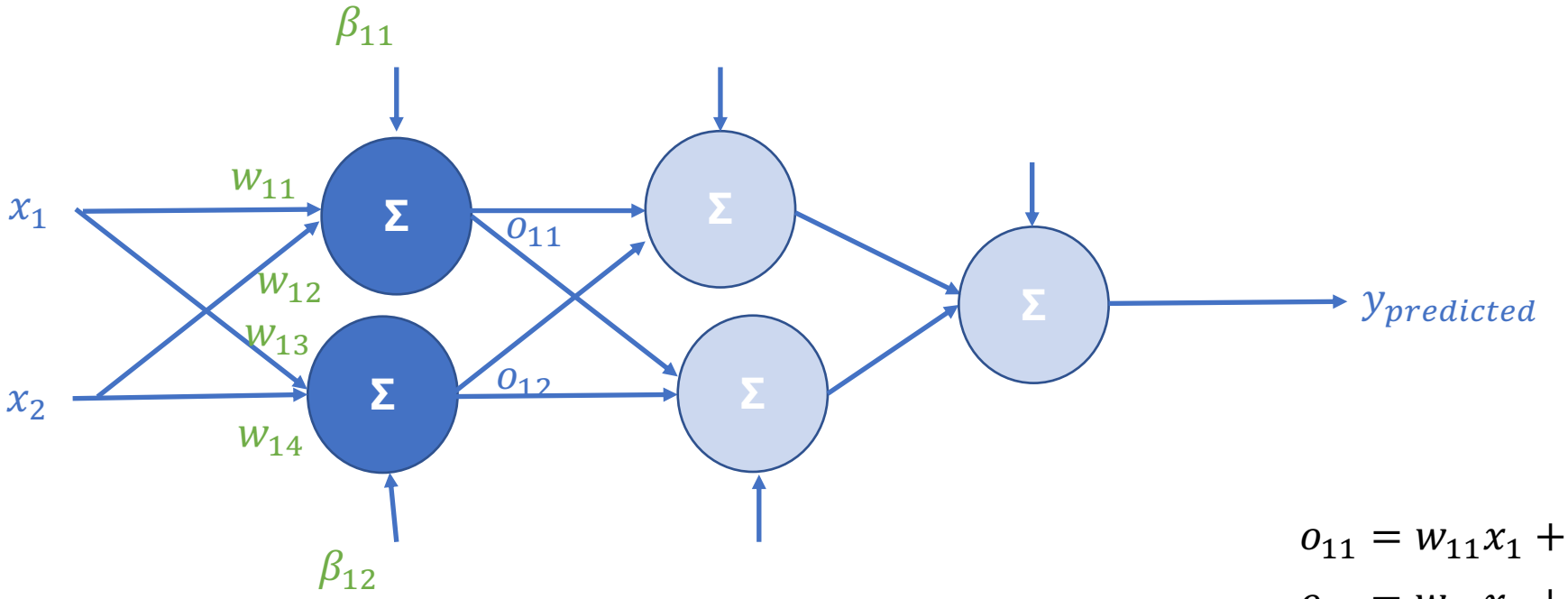
Gradient Descent in Feed Forward Nets

Let's consider this example network



Forward Pass (Compute $y_{predicted}$)

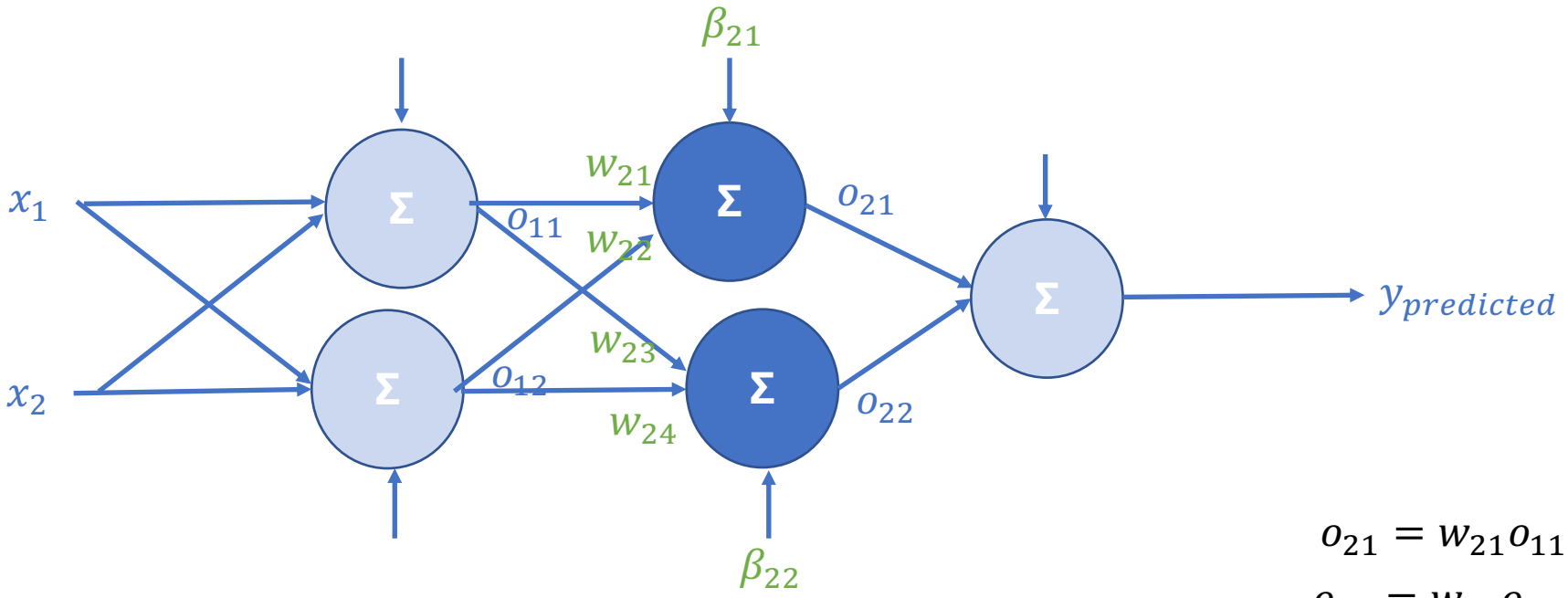
First, compute first layer output from x1, x2



$$o_{11} = w_{11}x_1 + w_{12}x_2 + \beta_{11}$$
$$o_{12} = w_{13}x_1 + w_{14}x_2 + \beta_{12}$$

Forward Pass (Compute $y_{predicted}$)

Second, compute second layer output from o11, o12

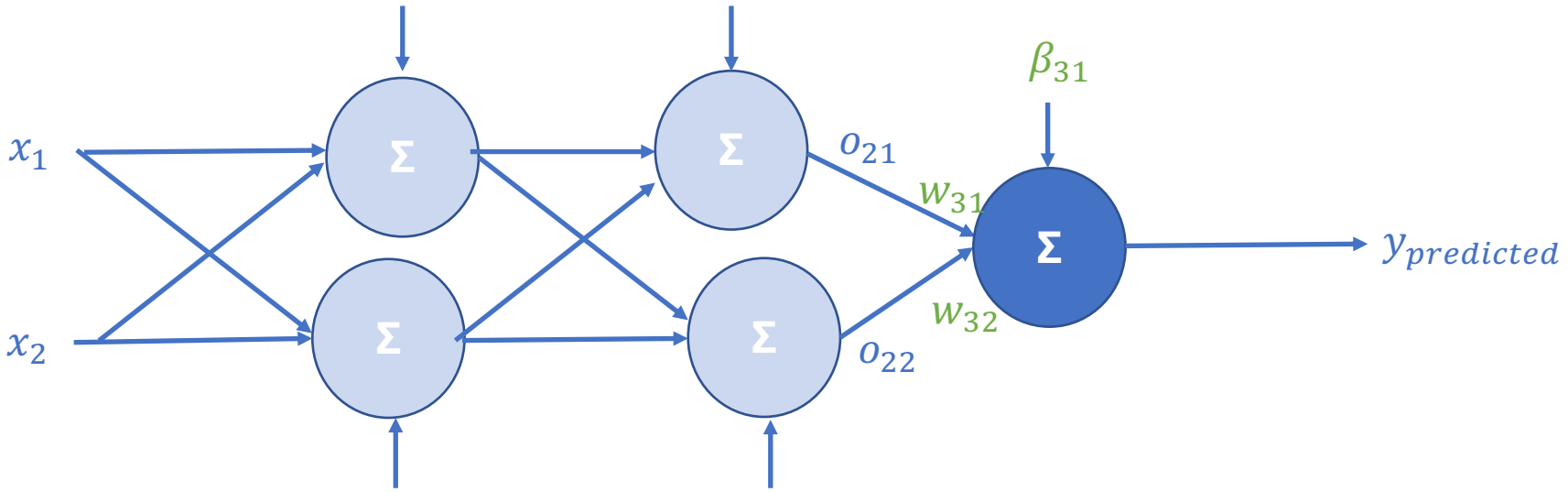


$$o_{21} = w_{21}o_{11} + w_{22}o_{12} + \beta_{21}$$

$$o_{22} = w_{23}o_{11} + w_{24}o_{12} + \beta_{22}$$

Forward Pass (Compute $y_{predicted}$)

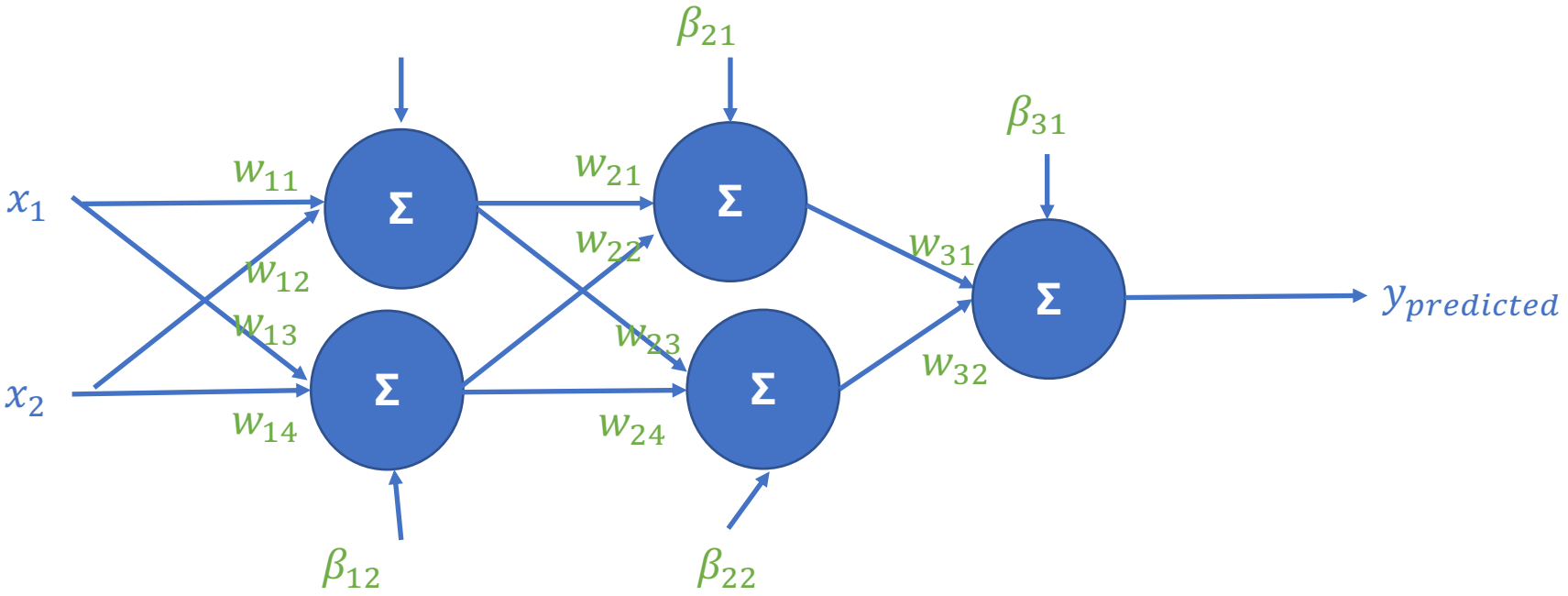
Third, compute y_predicted layer output from o21, o22



$$y_{predicted} = w_{31}o_{21} + w_{32}o_{22} + \beta_{31}$$

Forward Pass

Overall



$$y_{predicted}$$
$$= w_{31}(w_{21}(w_{11}x_1 + w_{12}x_2 + \beta_{11}) + w_{22}(w_{13}x_1 + w_{14}x_2 + \beta_{12}) + \beta_{21})$$
$$+ w_{32}(w_{23}(w_{11}x_1 + w_{12}x_2 + \beta_{11})) + w_{24}(w_{13}x_1 + w_{14}x_2 + \beta_{12}) + \beta_{22}) + \beta_{31}$$

Gradient Descent

- We need to compute gradient of error
- Considering MSE on M training data

$$Err = \frac{1}{M}\sum_{i=1}^{M}(y^i_{actual} - y^i_{predicted})2$$

$$= \frac{1}{M}\sum_{i=1}^{M}(y^i_{actual} - w_{31}(w_{21}(w_{11}x_1 + w_{12}x_2 + \beta_{11}) + w_{22}(w_{13}x_1 + w_{14}x_2 + \beta_{12}) + \beta_{21}) + w_{32}(w_{23}(w_{11}x_1 + w_{12}x_2 + \beta_{11})) + w_{24}(w_{13}x_1 + w_{14}x_2 + \beta_{12}) + \beta_{22}) + \beta_{31})2$$

And:

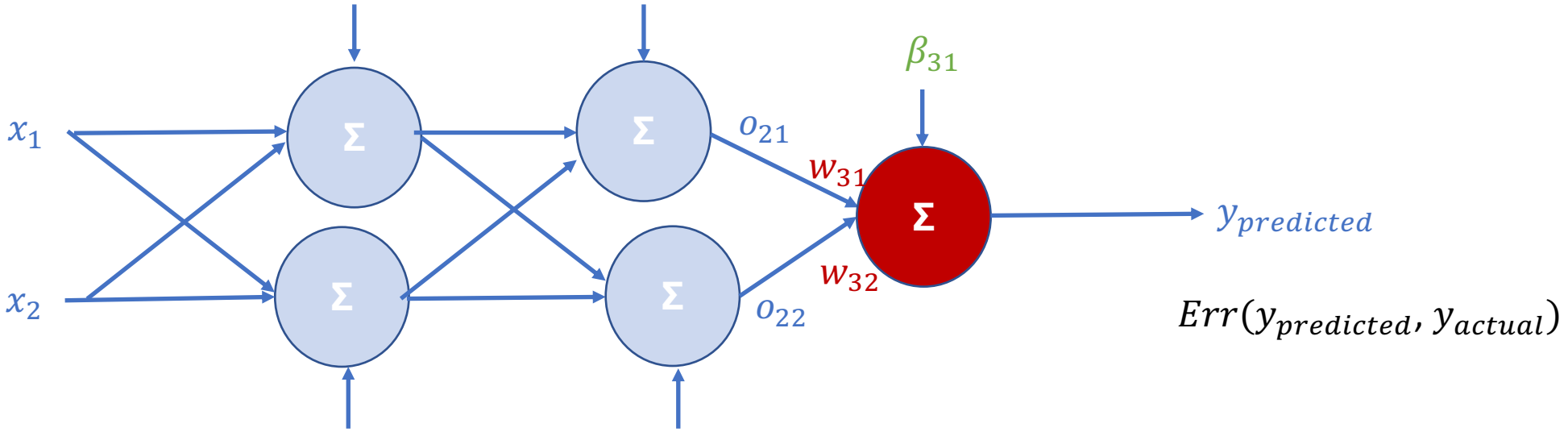
Each step in grad descent, we need $\nabla W = [\frac{\partial Err}{\partial w_{11}}, \frac{\partial Err}{\partial w_{12}}, \ldots, \frac{\partial Err}{\partial w_{21}}, \frac{\partial Err}{\partial w_{22}}, \frac{\partial Err}{\partial w_{23}}, \frac{\partial Err}{\partial w_{24}}, \frac{\partial Err}{\partial w_{31}}, \frac{\partial Err}{\partial w_{32}}, \frac{\partial Err}{\partial \beta_{11}}, \frac{\partial Err}{\partial \beta_{12,}}, \ldots]$

Solution ? Back Propagation of Gradients

- We can compute gradient of error with respect to input in each layer and “reuse” it for the computing gradients for the previous layer
- Error back propagates across layers

How?

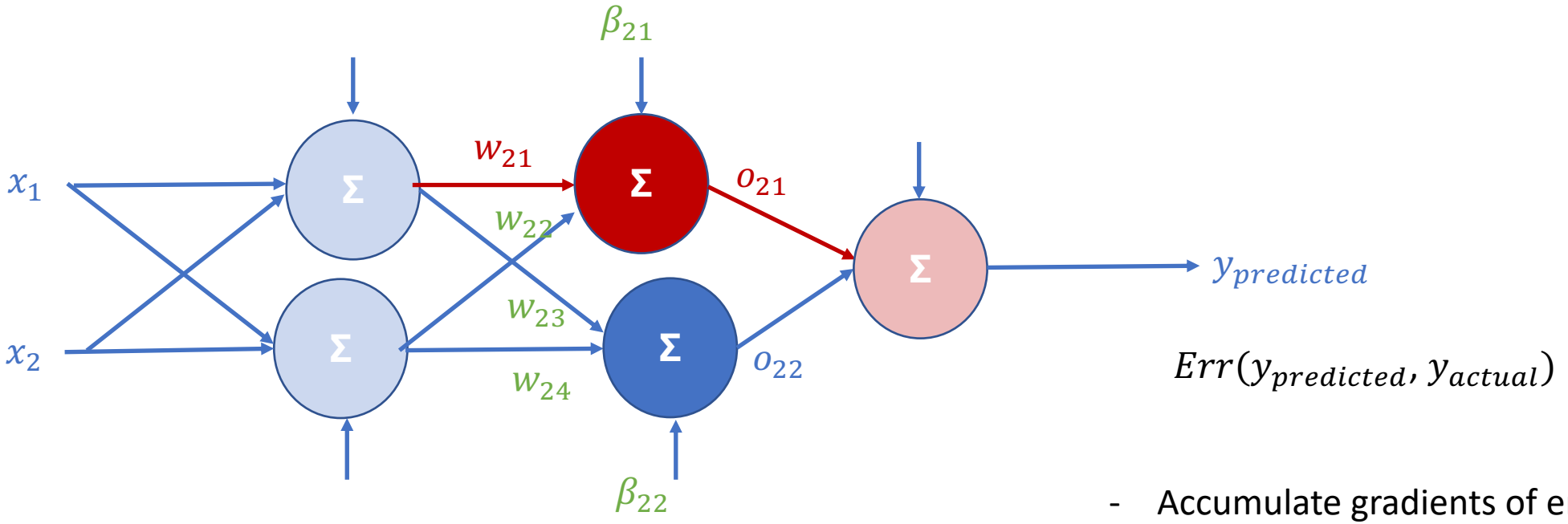
- Backword Pass (compute gradient of weights at last layer)
- Take only input o21 and o22 and forget about the previous layers



$Compute\ [\frac{\partial Err}{\partial w_{31}}, \frac{\partial Err}{\partial w_{32}}]$ in the same way we computed for a single perceptron

Then

- Backword Pass (compute gradient of weights at a previous layer)

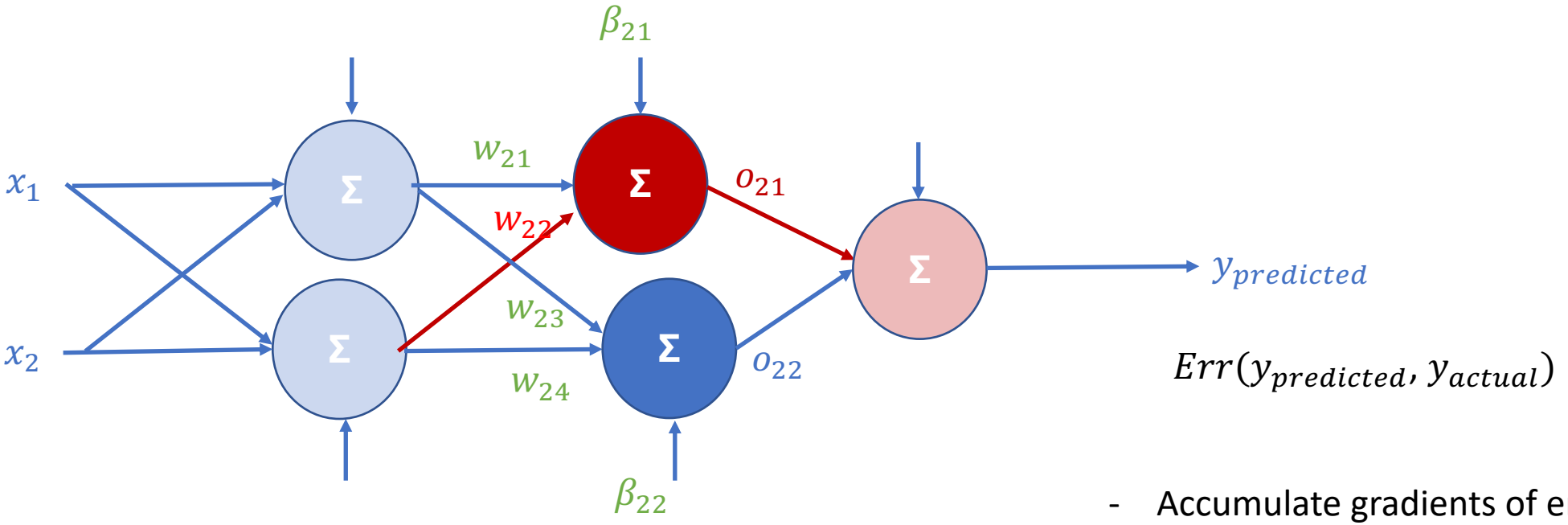


$$\frac{\partial Err}{\partial w_{21}} = \frac{\partial Err}{\partial o_{21}} \cdot \frac{\partial o_{21}}{\partial w_{21}}$$

- Accumulate gradients of error w.r.t inputs
- Compute gradients of immediate output w.r.t weights
- Multiply

Then

- Backword Pass (compute gradient of weights at a previous layer)



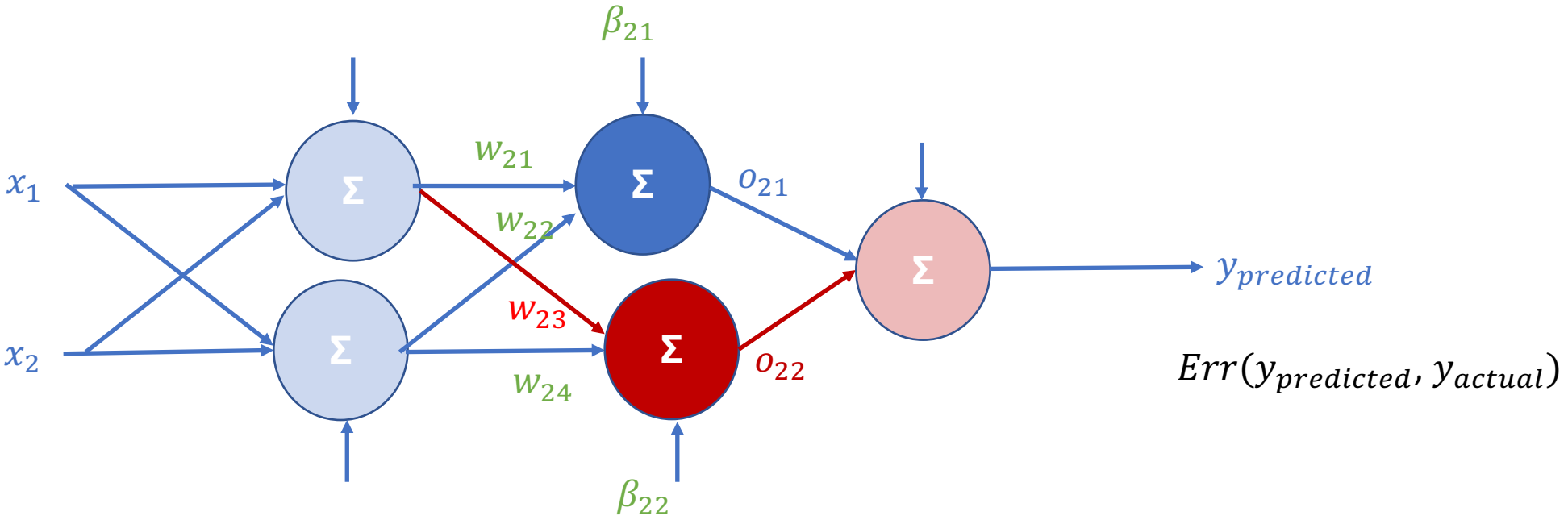
$$Err(y_{predicted}, y_{actual})$$

$$\frac{\partial Err}{\partial w_{22}} = \frac{\partial Err}{\partial o_{21}} . \frac{\partial o_{21}}{\partial w_{22}}$$

- Accumulate gradients of error w.r.t inputs
- Compute gradients of immediate output w.r.t weights
- Multiply

Then

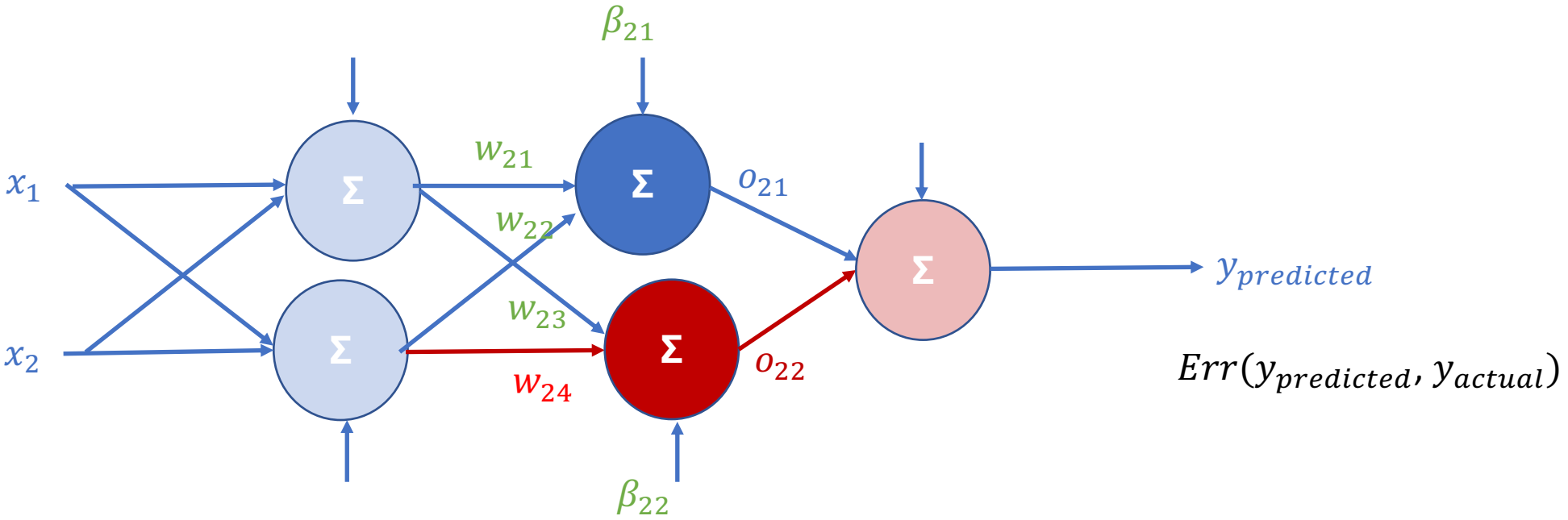
- Backword Pass (compute gradient of weights at a previous layer)



$$\frac{\partial Err}{\partial w_{23}} = \frac{\partial Err}{\partial o_{22}} \cdot \frac{\partial o_{22}}{\partial w_{23}}$$

Then

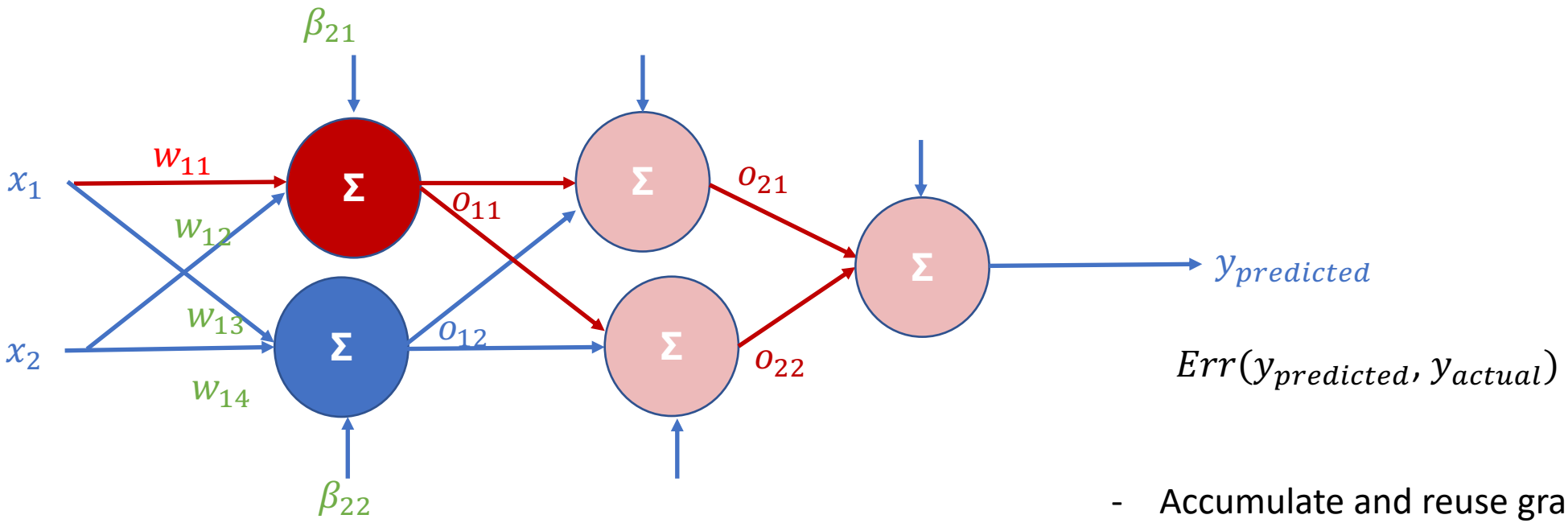
- Backword Pass (compute gradient of weights at a previous layer)



$$\frac{\partial Err}{\partial w_{24}} = \frac{\partial Err}{\partial o_{22}} \cdot \frac{\partial o_{22}}{\partial w_{24}}$$

Then

- Backword Pass (compute gradient of weights at first layer)

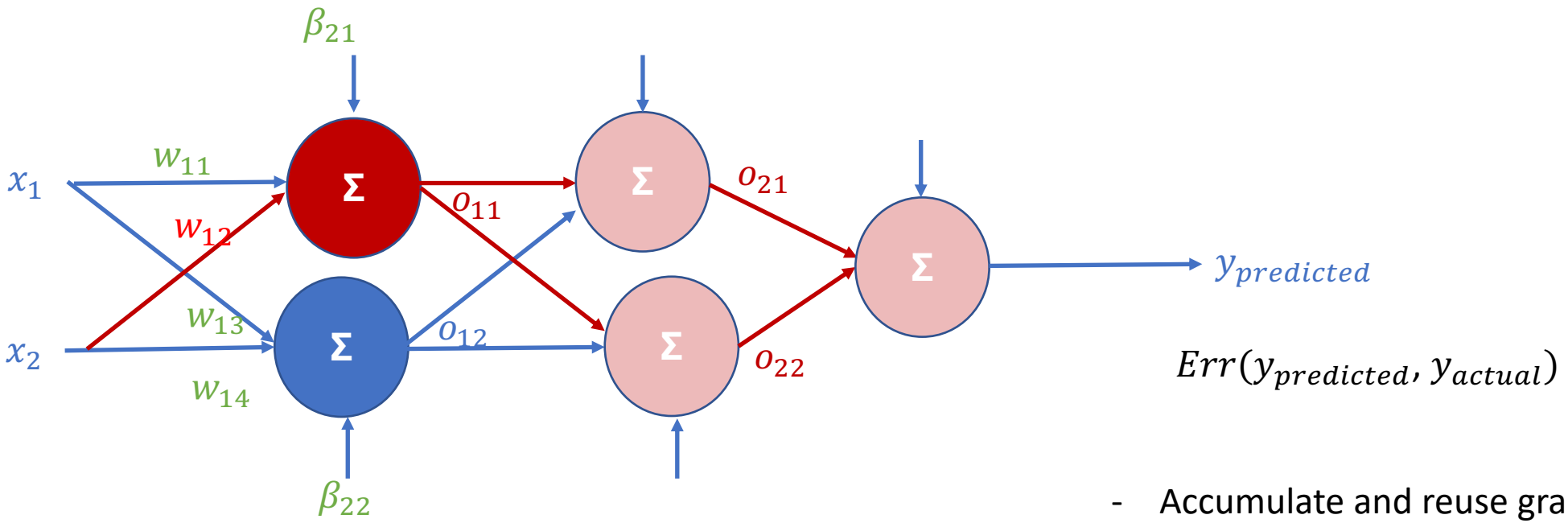


- Accumulate and reuse gradients of error w.r.t inputs

$$\frac{\partial Err}{\partial w_{11}} = \frac{\partial Err}{\partial o_{21}} . \frac{\partial o_{21}}{\partial o_{11}} . \frac{\partial o_{11}}{\partial w_{11}} + \frac{\partial Err}{\partial o_{22}} . \frac{\partial o_{22}}{\partial o_{11}} . \frac{\partial o_{11}}{\partial w_{11}} = \frac{\partial o_{11}}{\partial w_{11}} \left(\frac{\partial Err}{\partial o_{21}} . \frac{\partial o_{21}}{\partial o_{11}} + \frac{\partial Err}{\partial o_{22}} . \frac{\partial o_{22}}{\partial o_{11}} \right)$$

Then

- Backword Pass (compute gradient of weights at first layer)



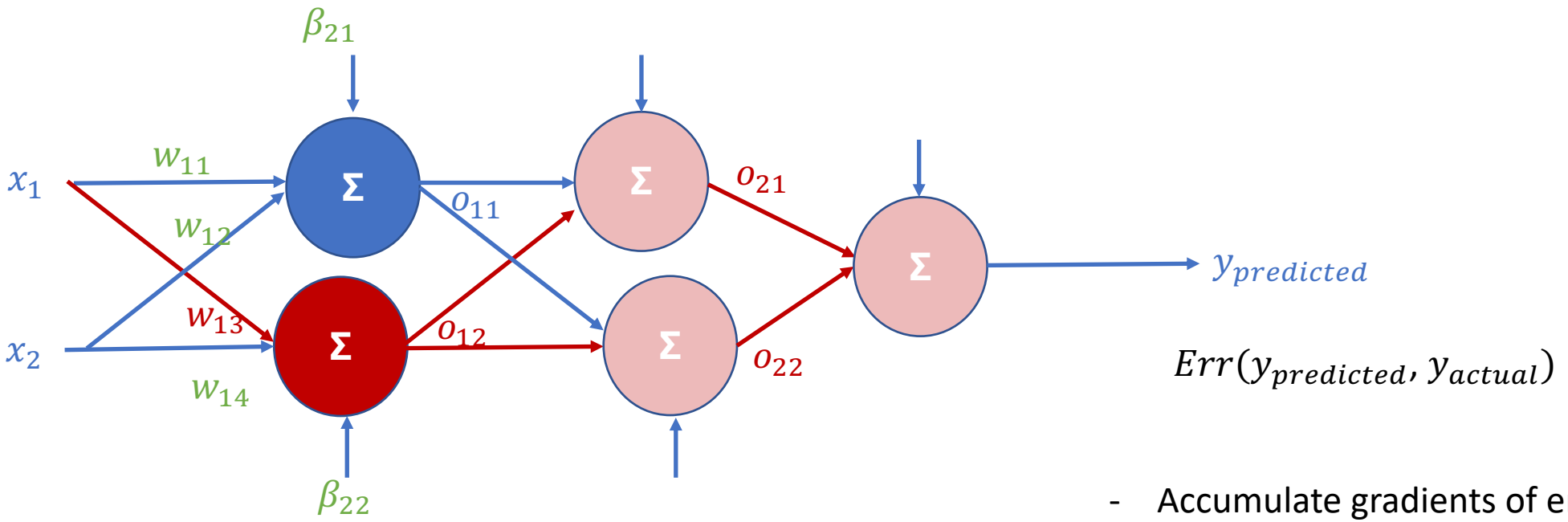
$Err(y_{predicted}, y_{actual})$

- Accumulate and reuse gradients of error w.r.t inputs

$$\frac{\partial Err}{\partial w_{12}} = \frac{\partial o_{11}}{\partial w_{12}}\left(\frac{\partial Err}{\partial o_{21}}.\frac{\partial o_{21}}{\partial o_{11}} + \frac{\partial Err}{\partial o_{22}}.\frac{\partial o_{22}}{\partial o_{11}}\right)$$

Then

- Backword Pass (compute gradient of weights at first layer)

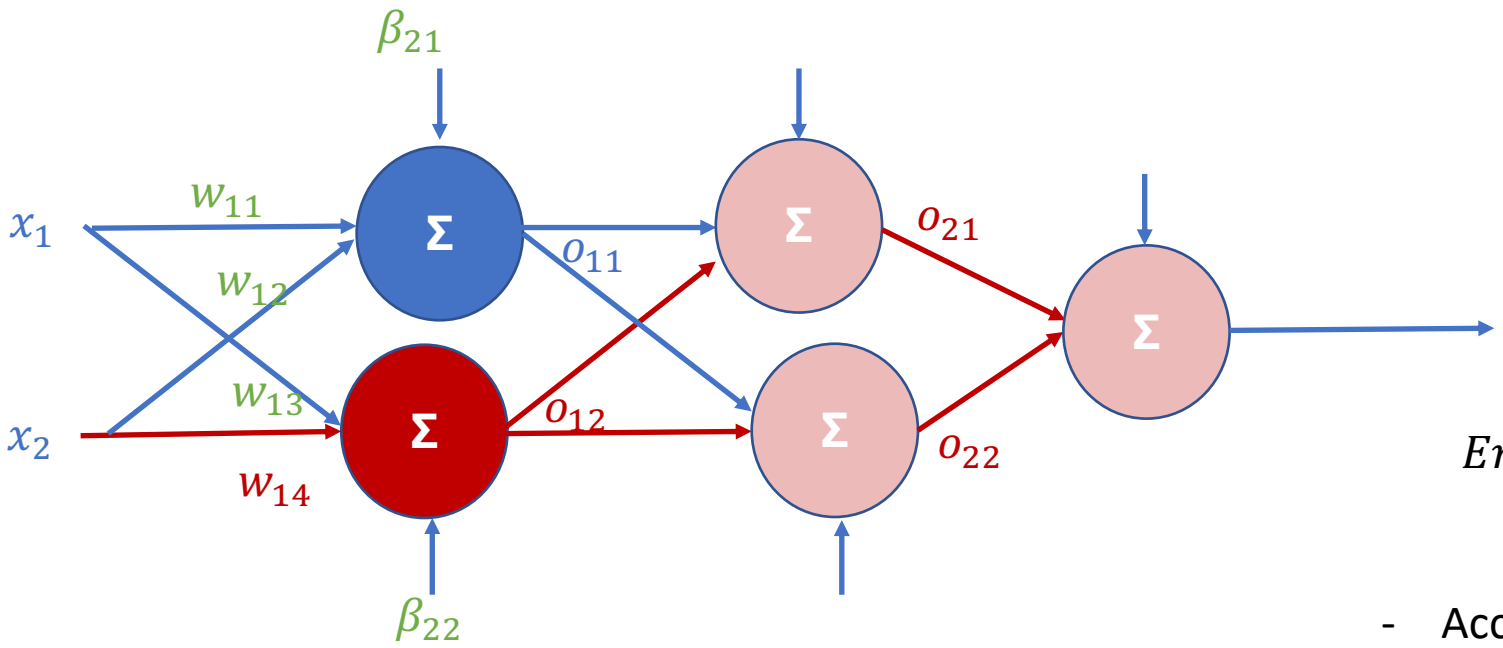


- Accumulate gradients of error w.r.t inputs

$$\frac{\partial Err}{\partial w_{13}} = \frac{\partial o_{12}}{\partial w_{13}}\left(\frac{\partial Err}{\partial o_{21}} \cdot \frac{\partial o_{21}}{\partial o_{12}} + \frac{\partial Err}{\partial o_{22}} \cdot \frac{\partial o_{22}}{\partial o_{12}}\right)$$

Then

- Backword Pass (compute gradient of weights at first layer)



$Err(y_{predicted}, y_{actual})$

- Accumulate gradients of error w.r.t inputs

$$\frac{\partial Err}{\partial w_{14}} = \frac{\partial o_{12}}{\partial w_{14}}\left(\frac{\partial Err}{\partial o_{21}} \cdot \frac{\partial o_{21}}{\partial o_{12}} + \frac{\partial Err}{\partial o_{22}} \cdot \frac{\partial o_{22}}{\partial o_{12}}\right)$$

What about the βs

- In similar ways as computing gradients for the Ws, we can compute gradients for the $\boldsymbol{\beta s}$ as well.

Training Strategy

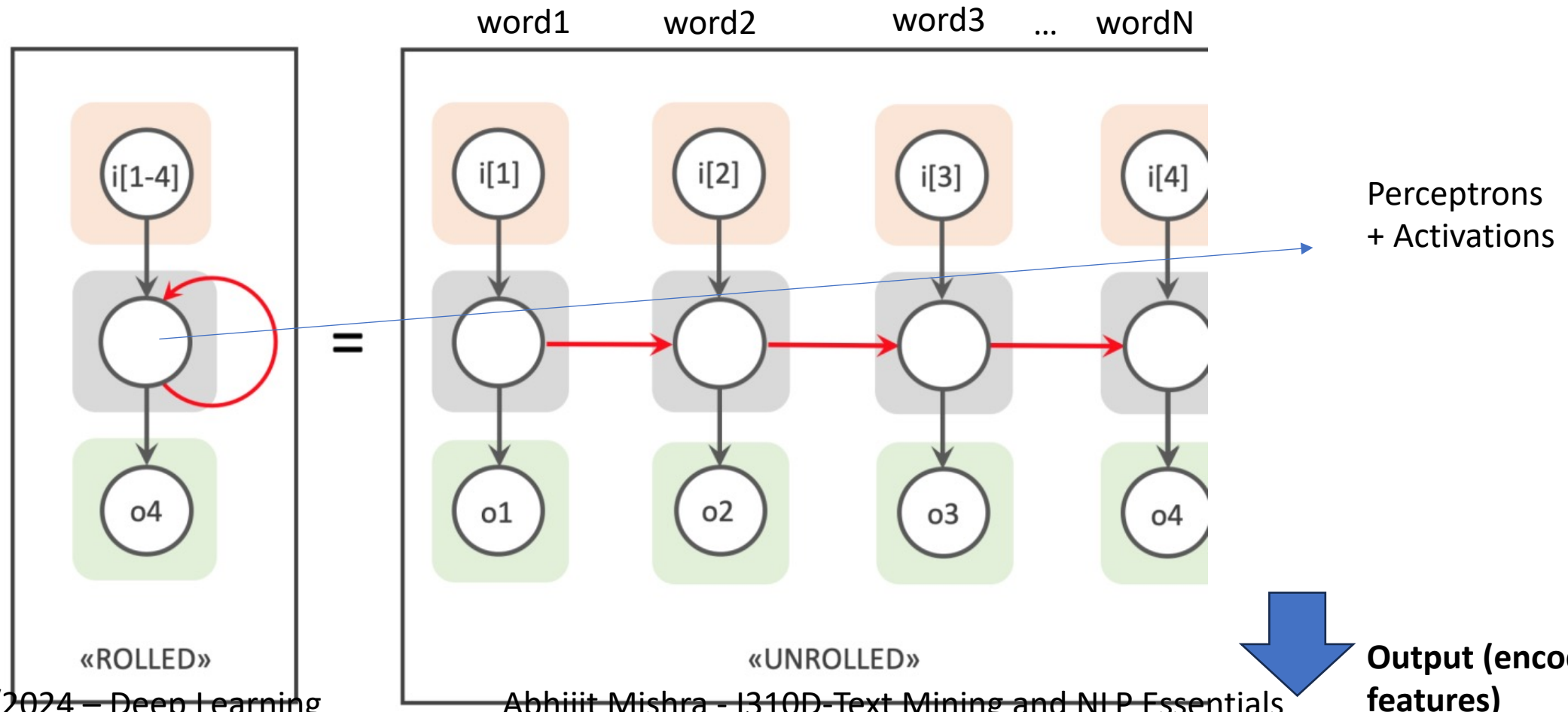
- Gradient Descent Requires computing Err on the whole data
- Computing Err on training data can be expensive (imagine 1M training examples)
  - Instead, we provide minibatches (e.g., batch_size = 16 or 16 training examples at a time)
  - Each mini batch comprises a set of randomly selected training examples
  - Gradient computed and back propagated for 1 mini-batch at a time
- We repeat training for all exclusive mini batches. This is called 1-epoch
- Typically training goes on for M epochs (say M=20)

https://distill.pub/2020/grand-tour/

https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=spiral&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.99619&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false

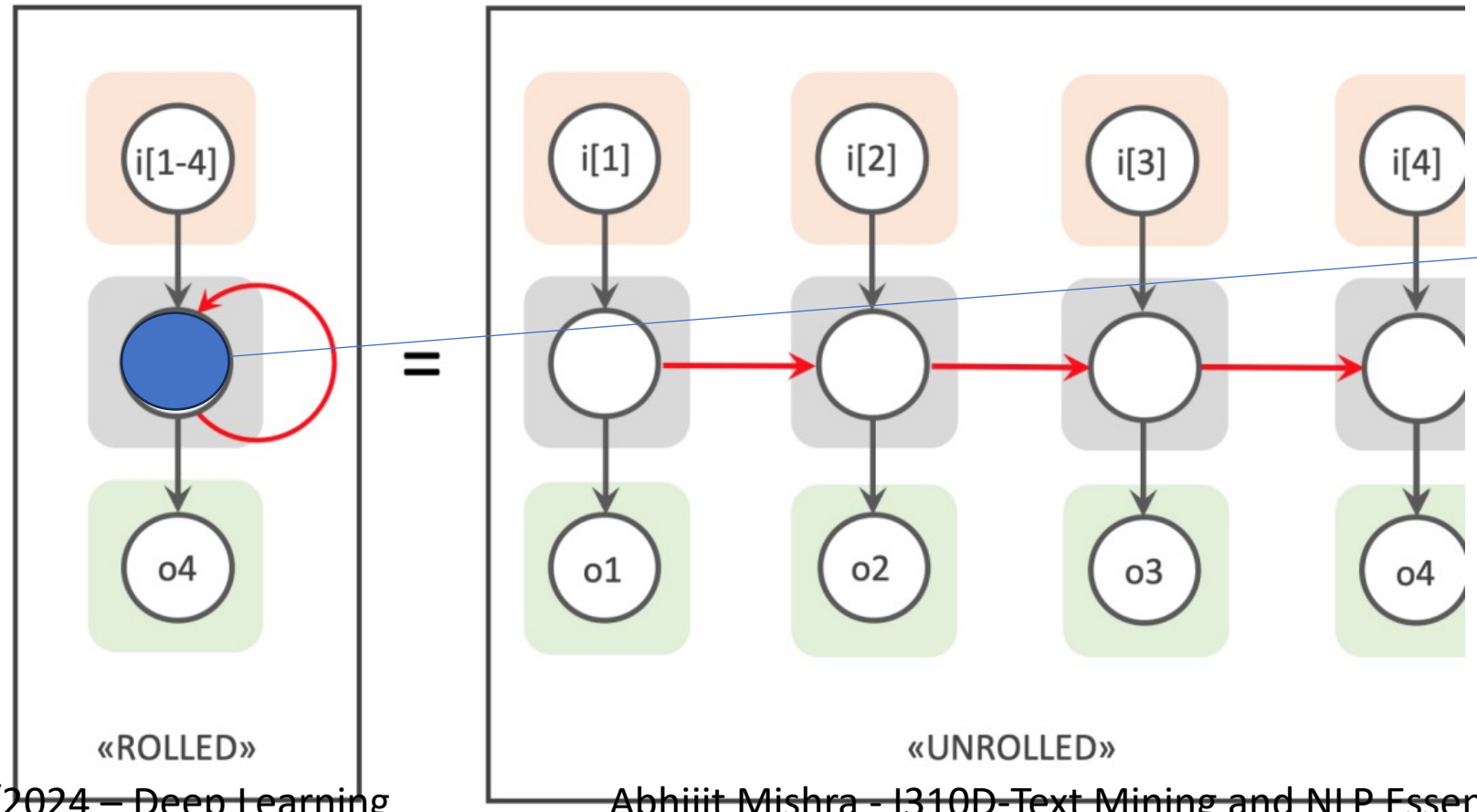
Some Advanced Neural Networks

Recurrent Neural Nets



Some Advanced Neural Networks

Long Short Term Memories

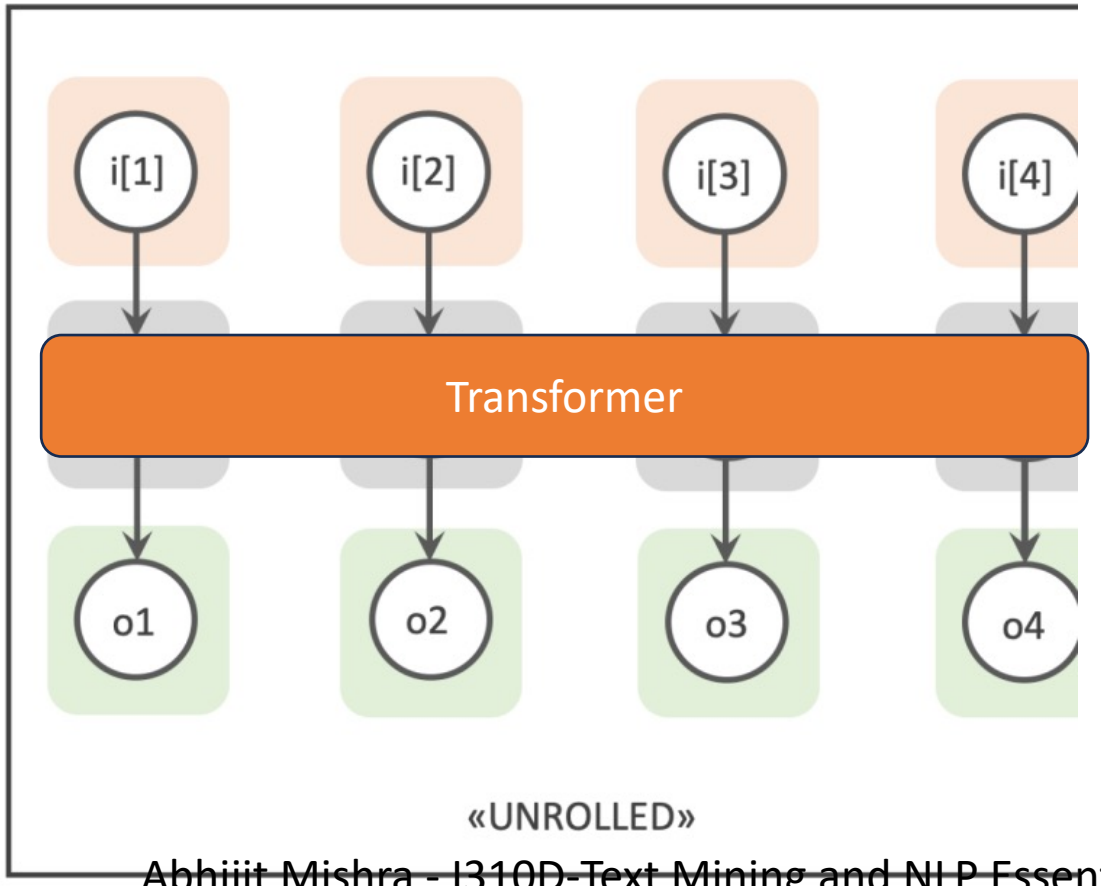


LSTM cells

Output (encoded features)

Some Advanced Neural Networks

Transformers



Output (encoded features)

Tasks solved using RNNs

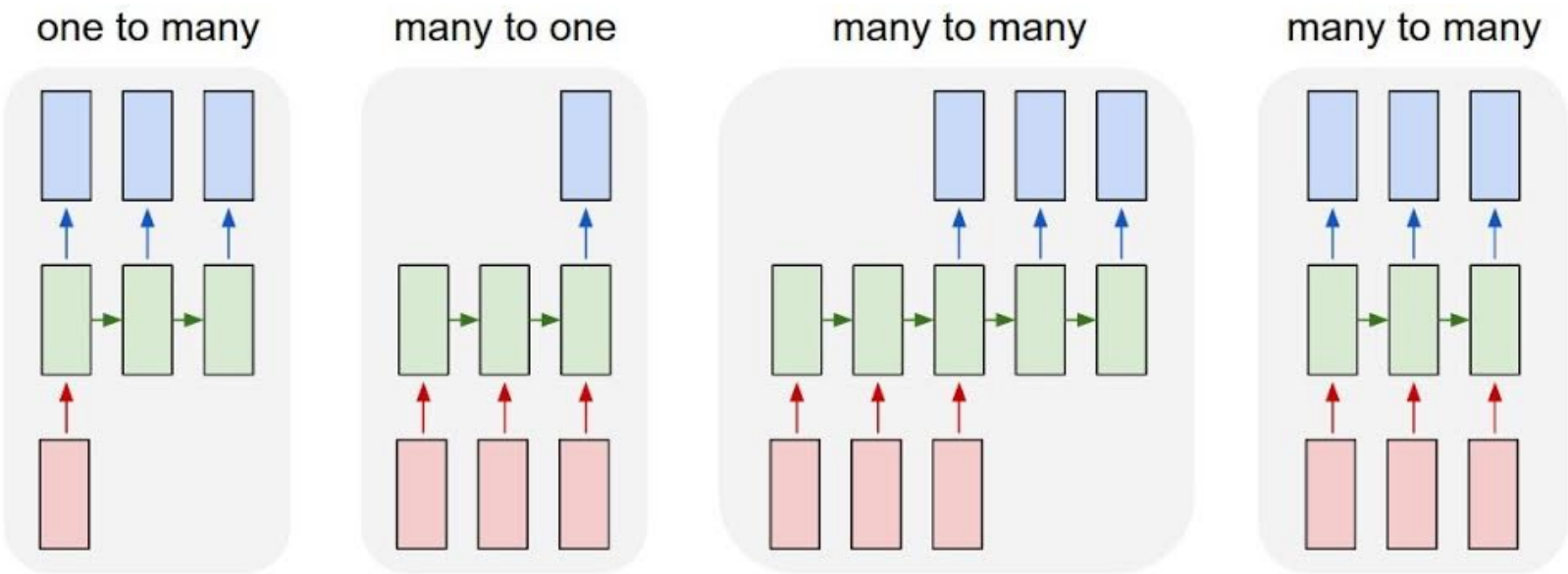


Image Captioning

Video Activity Recog / Text Classification

Video Captioning / Machine Translation

POS Tagging/ Language Modeling

Next Class

- Special Neural Nets – RNNs and Transformers basics
- Tutorial – Deep learning based classification