TEXAS
The University of Texas at Austin

I320D – Topics in Human Centered Data Science

Text Mining and NLP Essentials

Week 11: Deep Learning for NLP II, Back Propagation, RNNs and Transformers, Language Models and Word Wmbeddings

Dr. Abhijit Mishra

Now: In class quiz (participation)

- https://utexas.instructure.com/courses/1382133/quizzes/1893214
- 10 MCQs: 10 mins
- Full points for in-class participation
- Time limit: 10 mins

Week 10-11 Activities

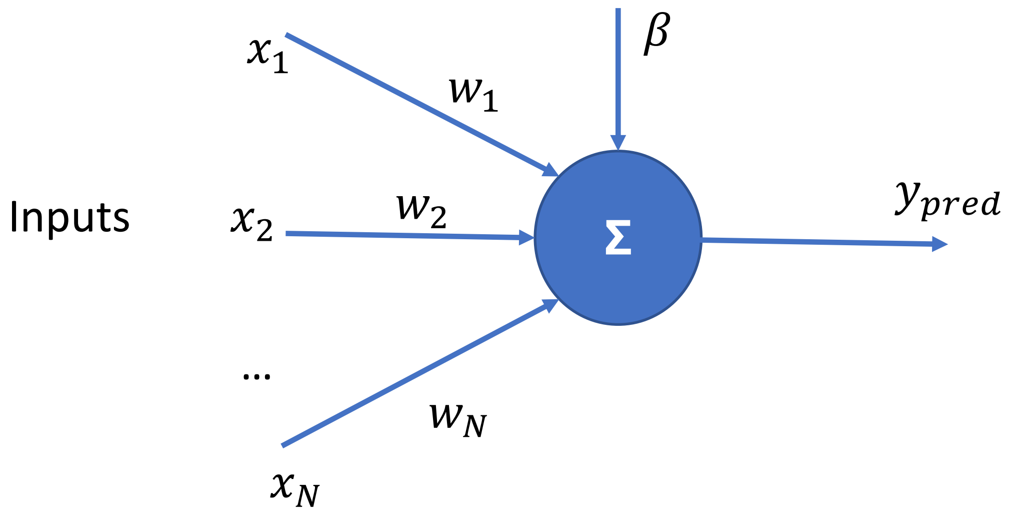
- Project Proposals Due **Today** (**03/26/2023**):
  - Maximum 2 pages – one submission per group
  - https://utexas.instructure.com/courses/1382133/assignments/6619554?module_item_id=13585860
  - Sample project reports: Canvas->Files->Sample_project_reports
- Assignment 5: (last take home assignment)
  - Will be based on fine-tuning of a pre-trained language model (Lab on Thursday)

Last Week

- Introduction to Deep Learning for NLP
- Perceptrons and Feed Forward Networks
- Gradient Descent Algorithm

Recap: Perceptrons

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



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

OR

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

$$\Theta = [\mathrm{w}_1, w_2, \dots, w_N]$$

Recap: Training a perceptron

Training with linear parametric decision functions

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

- **In this case**

$$\underset{w_1, w_2, \beta}{\text{argmin}} \sum_{i=1}^{M} \left(y_{actual}^{i} - (w_1 x_1 + w_2 x_2 + \beta)\right)^2$$

Recap: 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

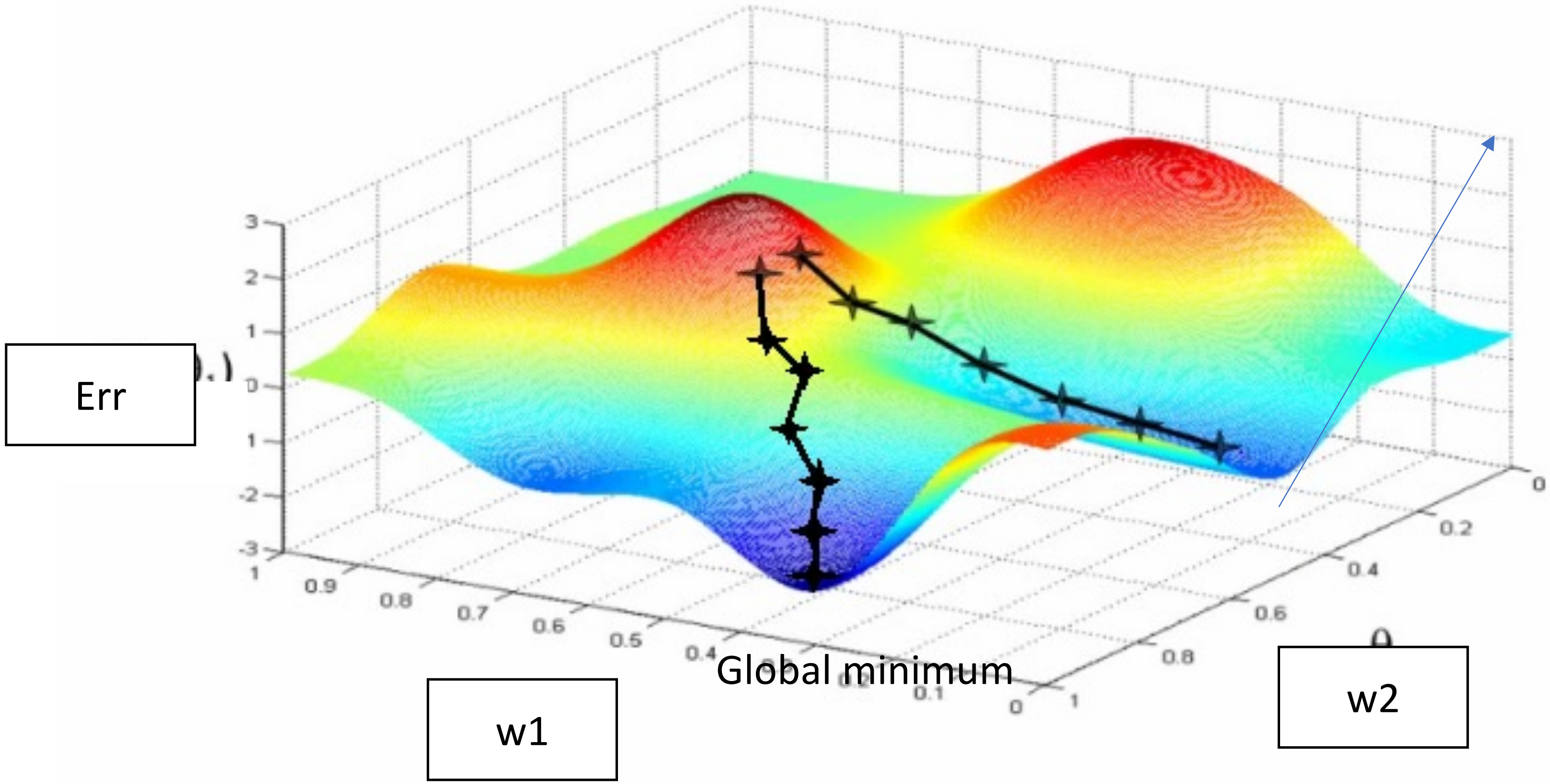
$\boldsymbol{\eta} \rightarrow$ *A user defined paramter (a.k.a Hyperparameter) also called as "learning rate"*

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

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

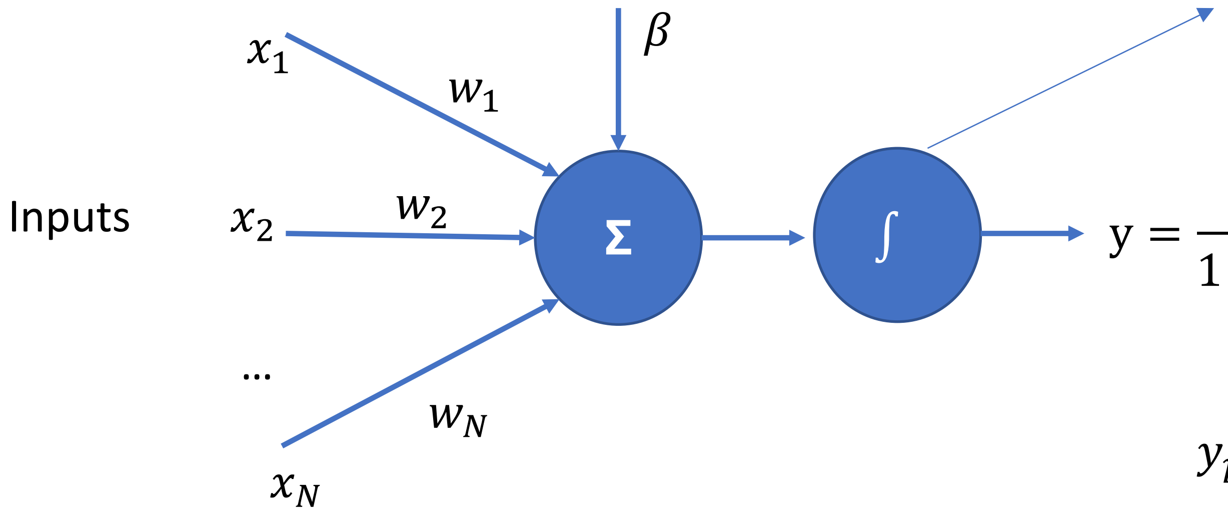
Recap: Gradient Descent: Intuition (...)

- Consider simple Err function with two parameters w1, w2



Recap: Can we make the perceptrons non-linear? – Activation Functions

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



Non linear activation

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

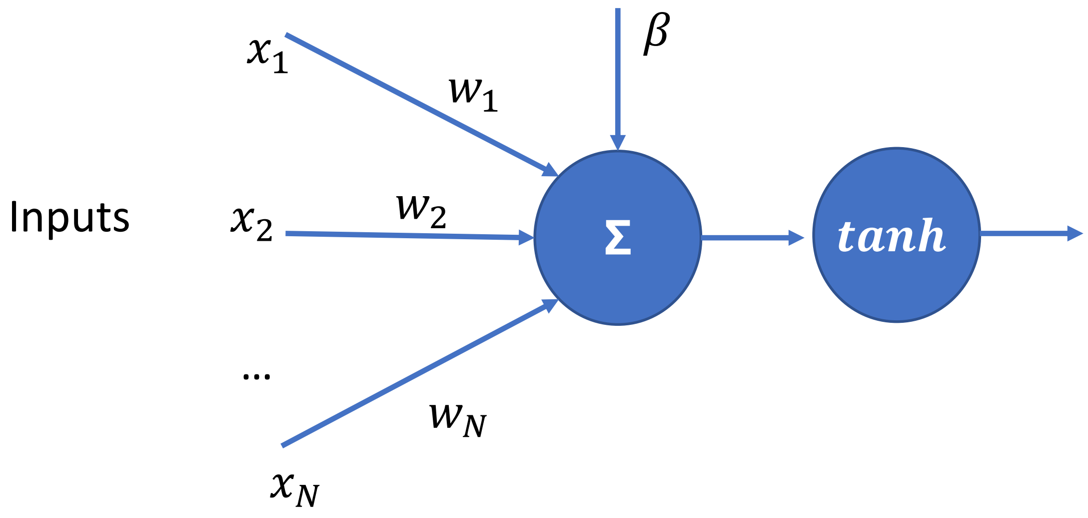
$$y_{pred} = \begin{cases} 1\ if\ y > 0.5 \\ 0\ if\ y \leq 0.5 \end{cases}$$

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

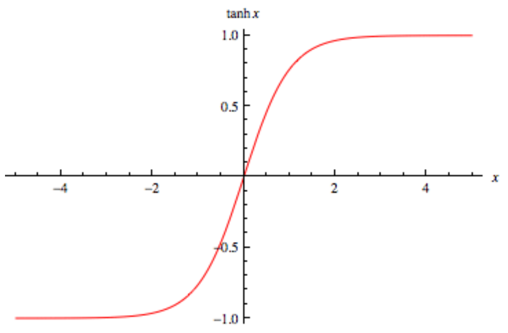
Logistic Regression (classification)

Activation: TanH

- 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

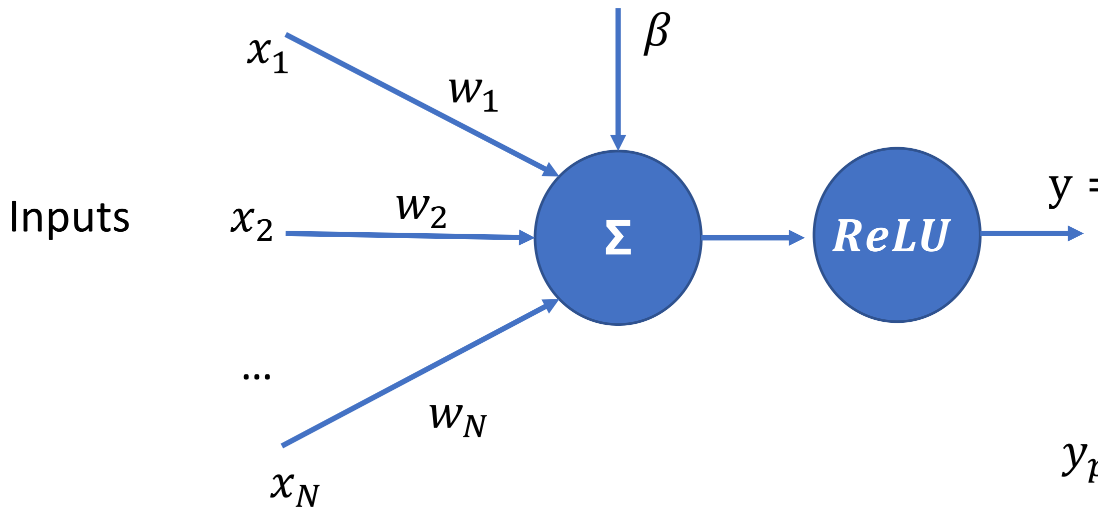
$$y_{pred} = \begin{cases} 1 \; if \; y > 0 \\ -1 \; if \; y \leq 0 \end{cases}$$

Which model is this? classification or regression?

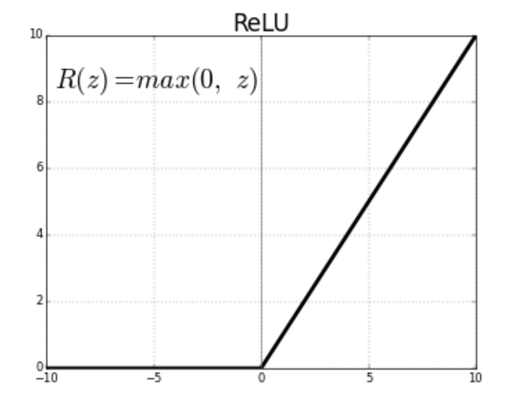
Classification

Activation: ReLU

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



$$y = \max(0, W^T X + \beta)$$



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

Which model is this? classification or regression?

Classification + Regression ?

Recap: Neural Networks

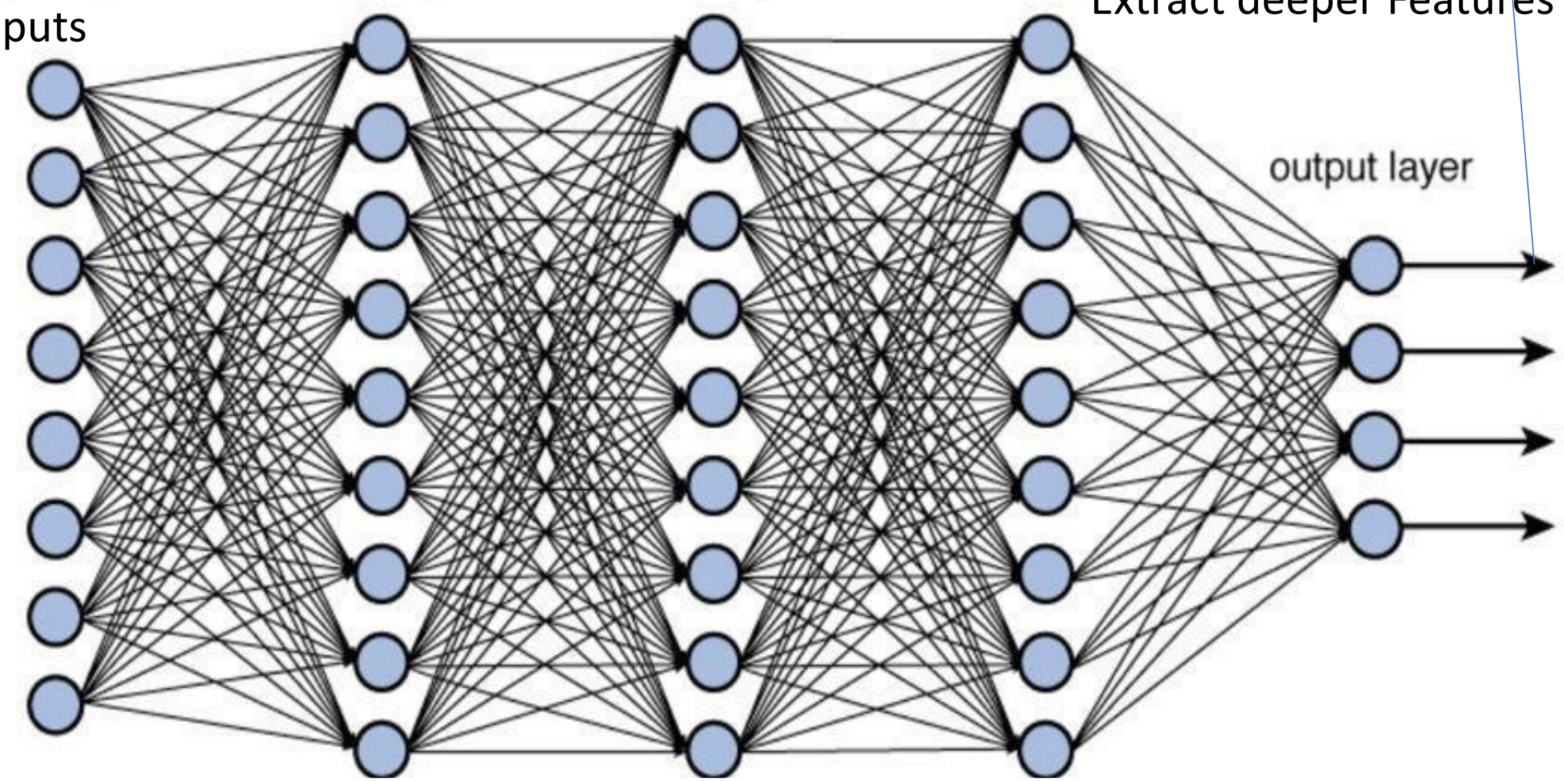
Form a artificial neural network by stacking many perceptron elements with / without activation functions

The mathematical 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>

Text Classification example

I -> [0, 1, 0, 0, …]

love -> [0, 0, 1, 0, …]

this -> [0, 0, 0, 1, …]

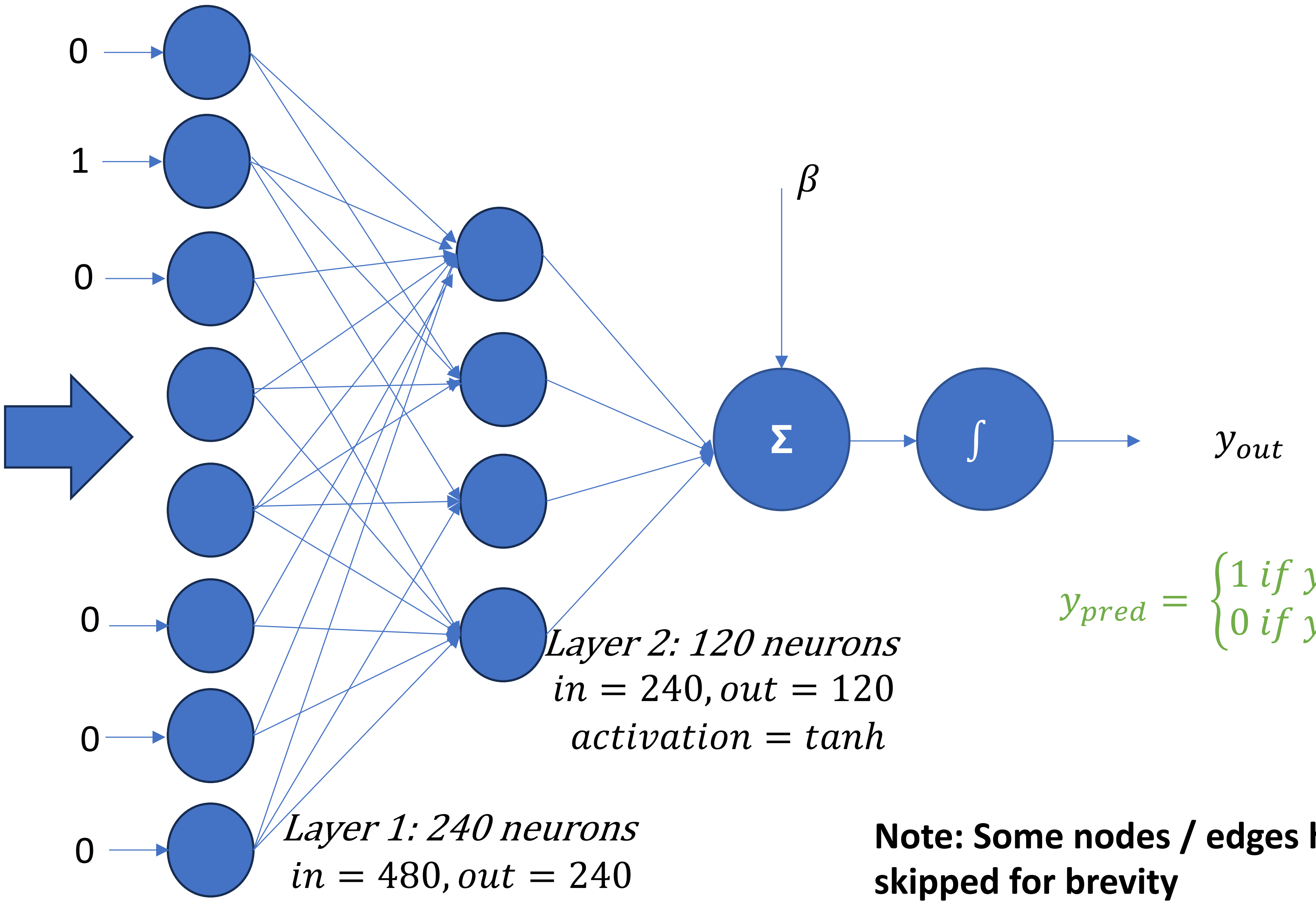
product [0, 0, 0, 0, …]

<PAD> [1, 0, 0, 0]

<PAD> [1, 0, 0, 0]

…

Size = 10*48 = 480

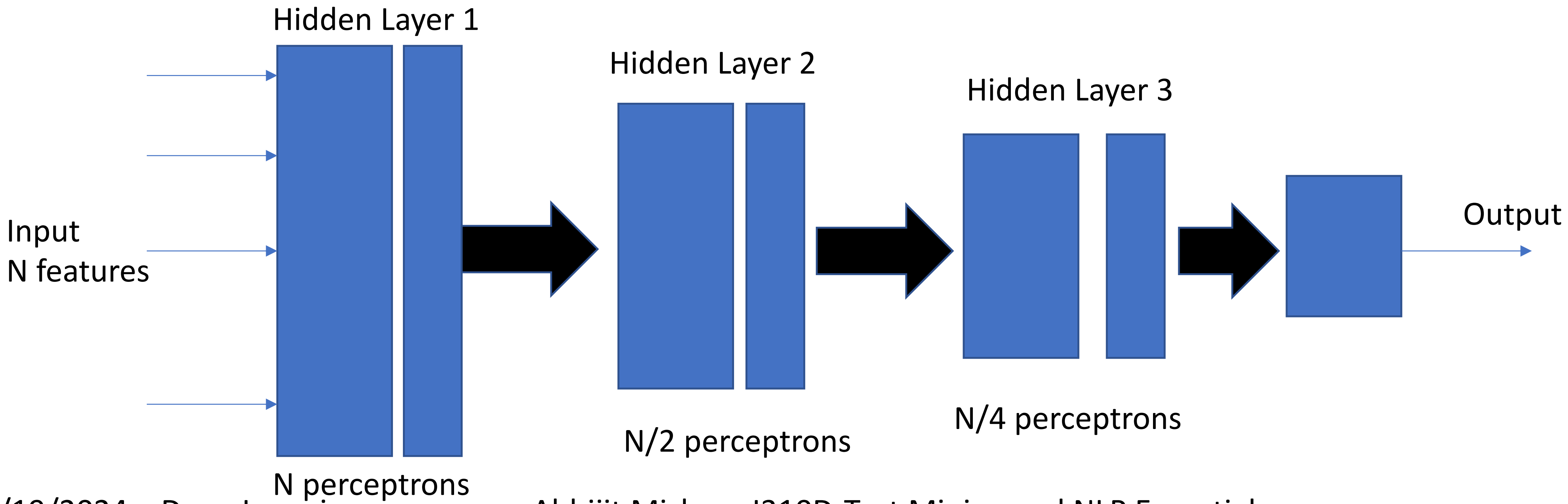


$$y_{pred} = \begin{cases} 1\ if\ y_{out} > 0.5 \\ 0\ if\ y_{out} \leq 0.5 \end{cases}$$

Note: Some nodes / edges have been skipped for brevity

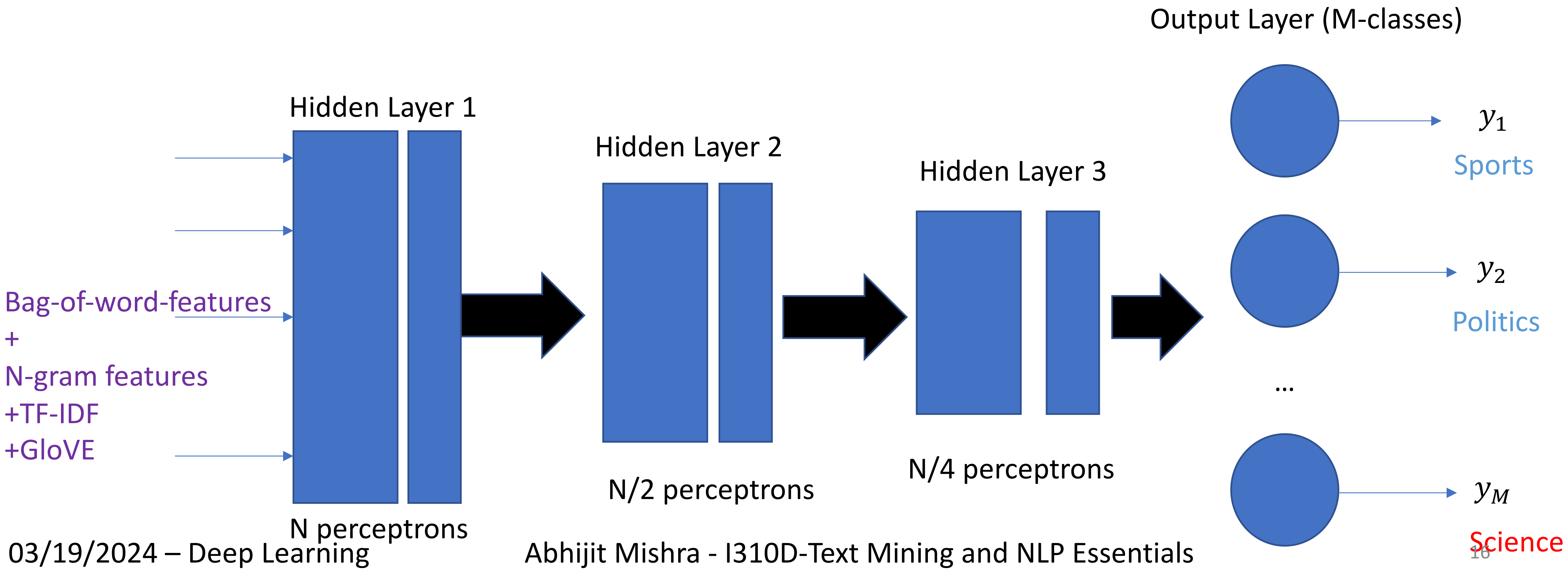
Recap: Feed forward neural network

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



Recap: Feed Forward Network: Text Classification Example

- M-class classification



In Python - Binary Classification Example

- `model = Sequential()`
- `model.add(Dense(16, input_shape=(vocab_size,), activation='relu')) # 16 units in the hidden layer`
- `model.add(Dense(8))`
- `model.add(Dense(1, activation='sigmoid')) # Output layer with sigmoid activation for binary classification`

Week 11: Roadmap

- Training a neural network
  - Back propagation basics
- Advanced neural networks for text modeling
  - Recurrent Neural Networks
  - Transformers
- Language Models
  - Achieving Different Language Modeling objectives with Deep Neural Architectures

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

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

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

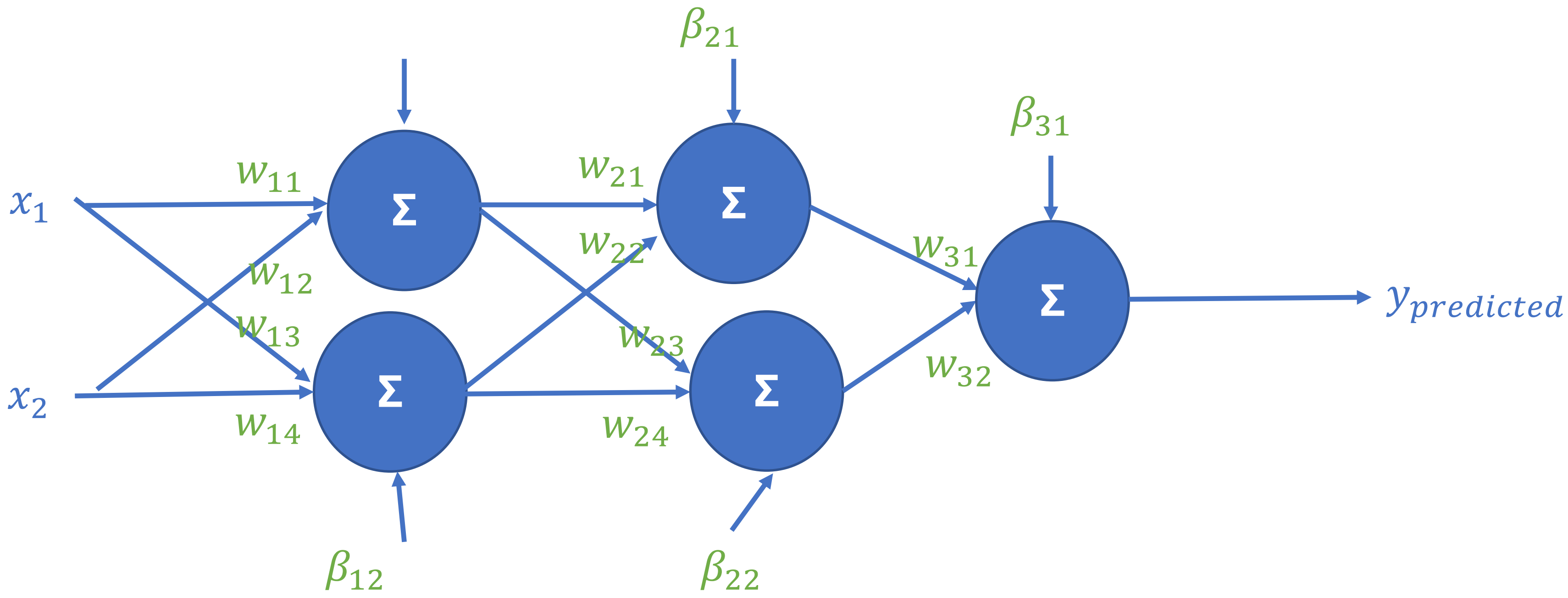
$$\ldots$$

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

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

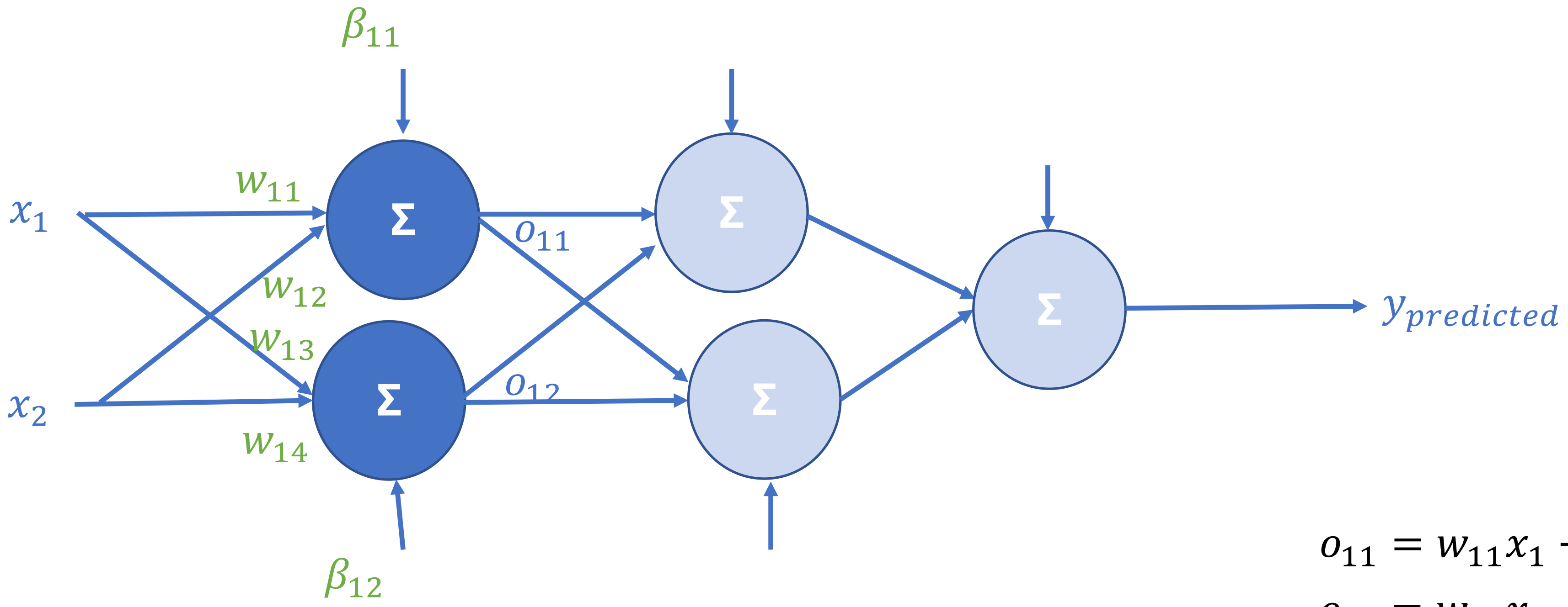
Gradient Descent in Feed Forward Nets

Let's consider this example network



Forward Pass (Compute $y_{predicted}$)

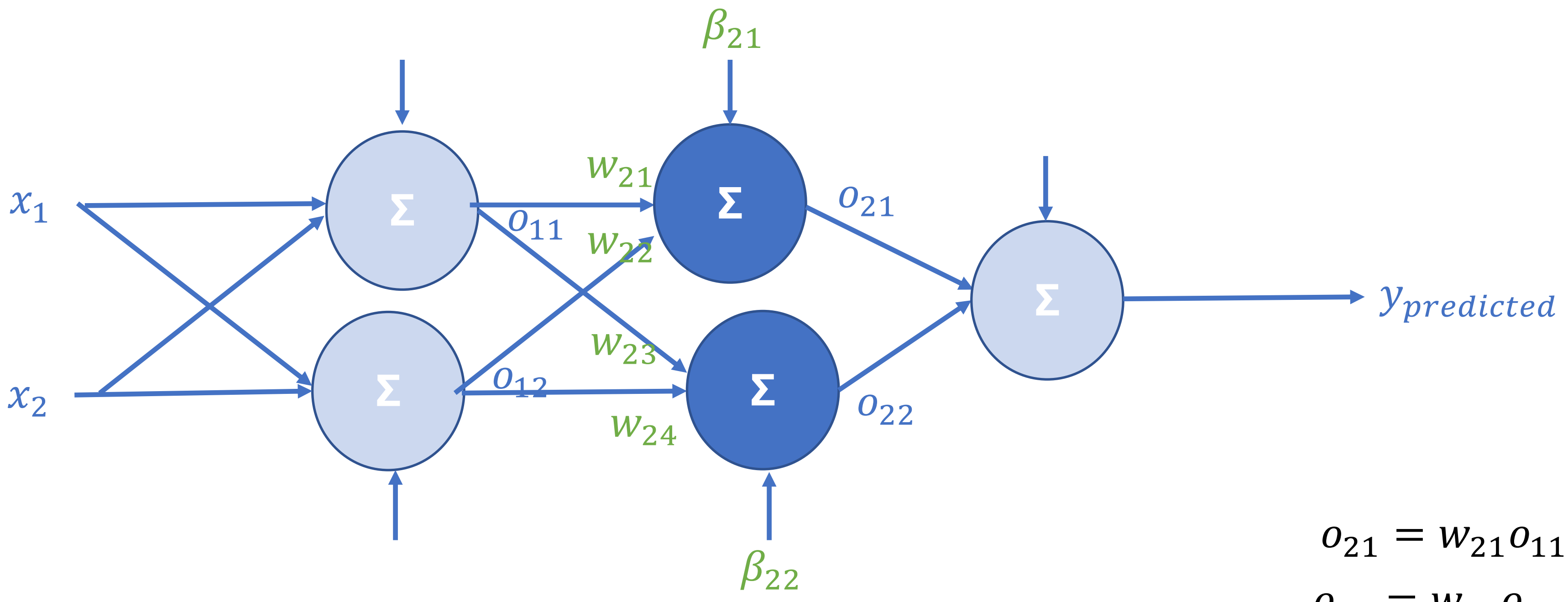
First, compute first layer output from x1, x2



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

Forward Pass (Compute $y_{predicted}$)

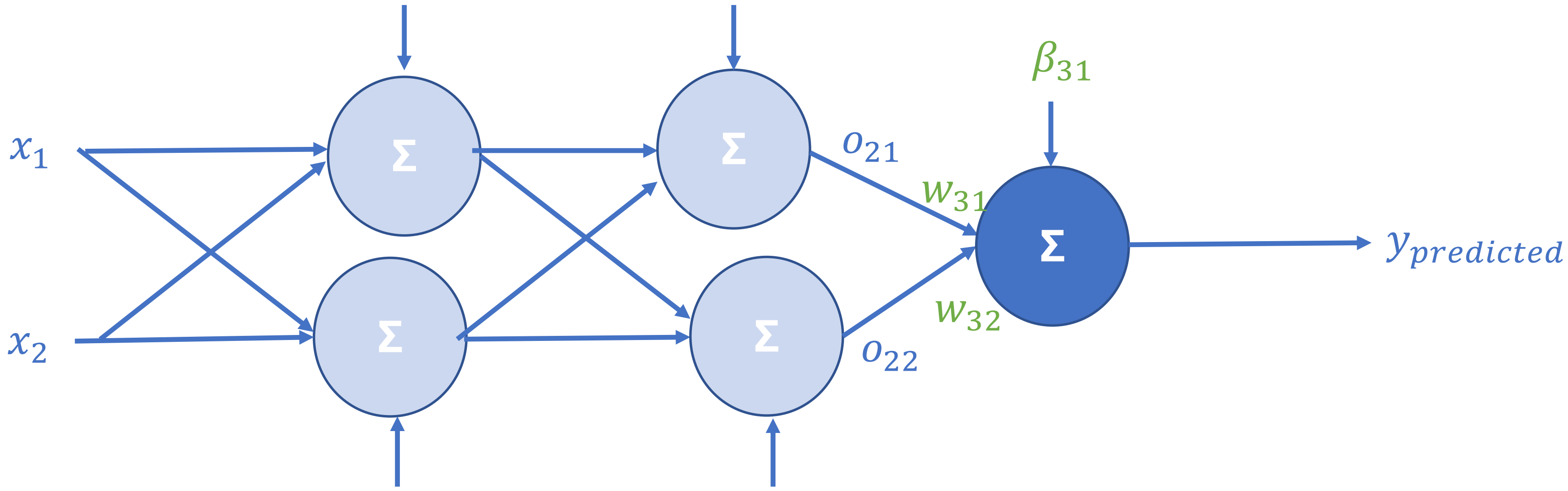
Second, compute second layer output from o11, o12



$$o_{21} = w_{21}o_{11} + w_{22}o_{12} + \beta_{21}$$
$$o_{22} = w_{23}o_{11} + w_{24}o_{12} + \beta_{22}$$

Forward Pass (Compute $y_{predicted}$)

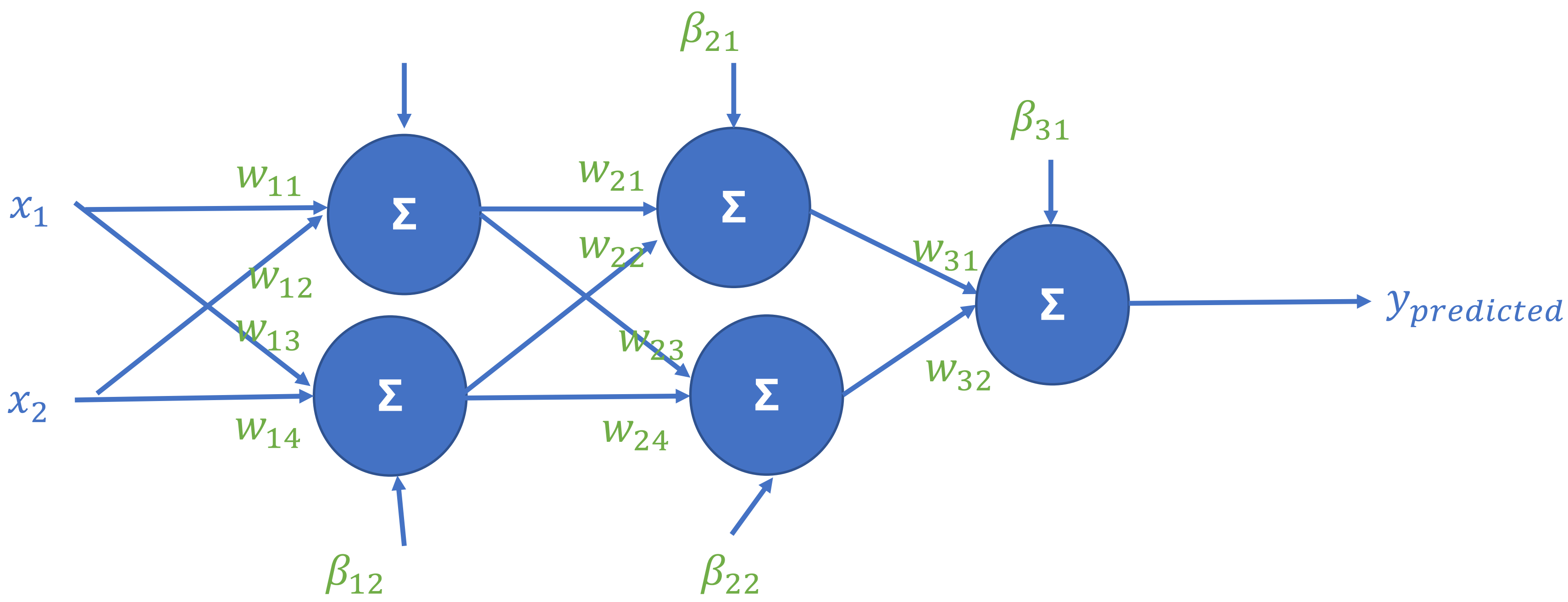
Third, compute y_predicted layer output from o21, o22



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

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 with Feed Forward Nets

- We need to compute gradient of error
- Considering Error on M training data (take MSE for example)

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

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

And:

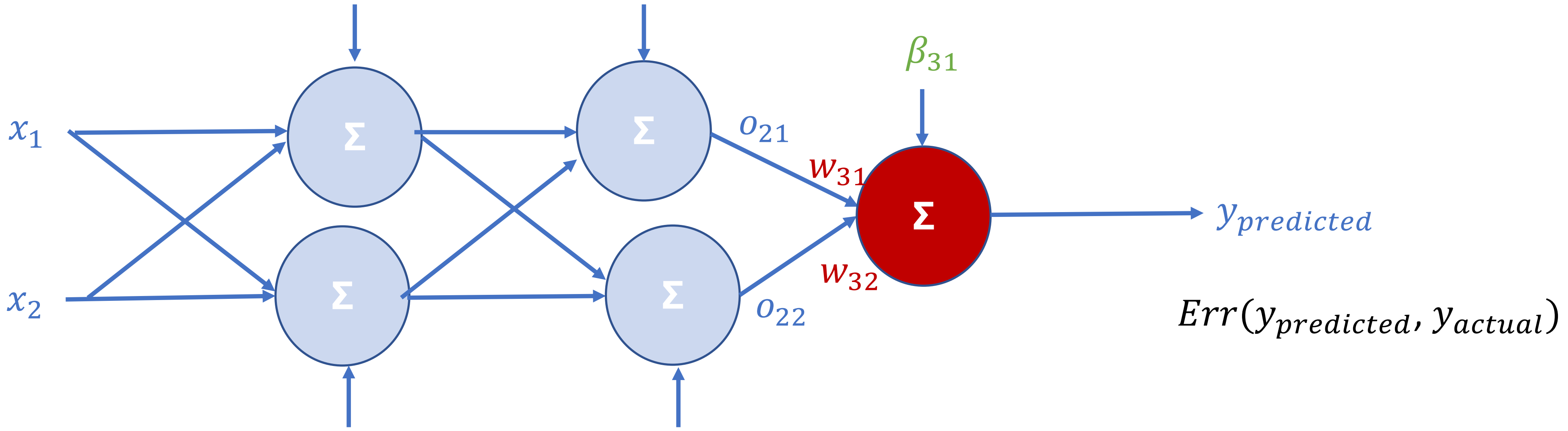
Each step in grad descent, we need $\nabla W = [\frac{\partial Err}{\partial w_{11}}, \frac{\partial Err}{\partial w_{12}}, \ldots, \frac{\partial Err}{\partial w_{21}}, \frac{\partial Err}{\partial w_{22}}, \frac{\partial Err}{\partial w_{23}}, \frac{\partial Err}{\partial w_{24}}, \frac{\partial Err}{\partial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

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?

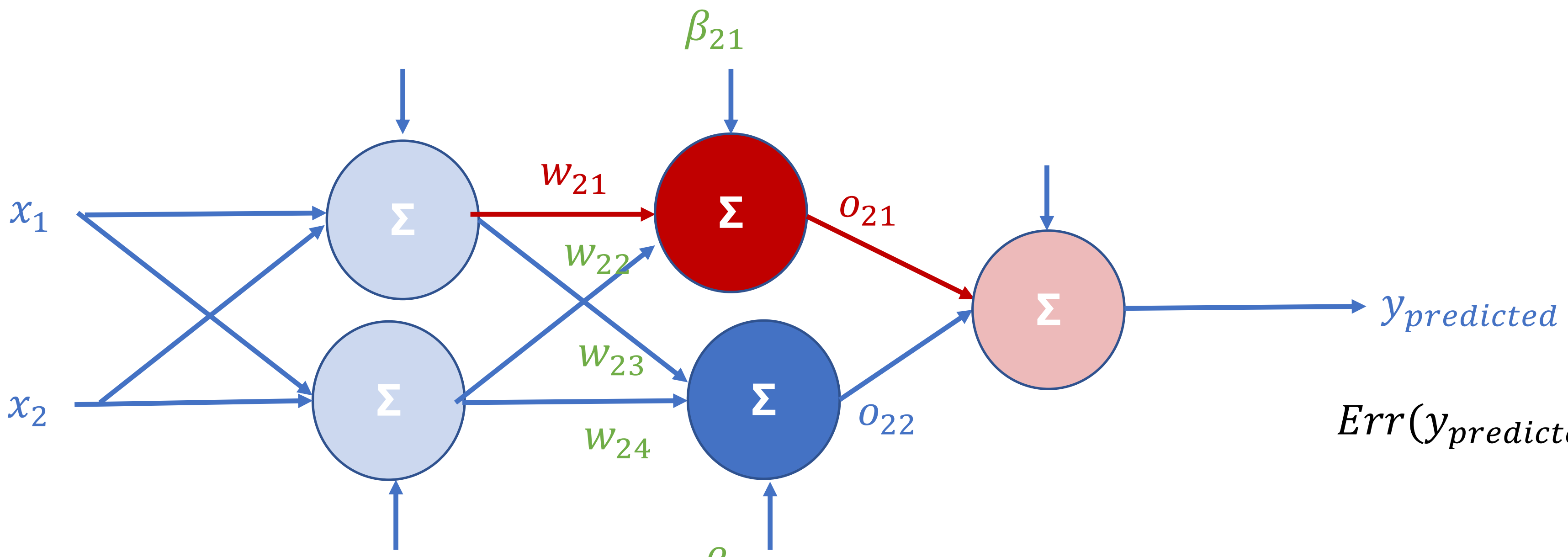
- Backword Pass (compute gradient of weights at last layer)
- Take only input o21 and o22 and forget about the previous layers



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

Then

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



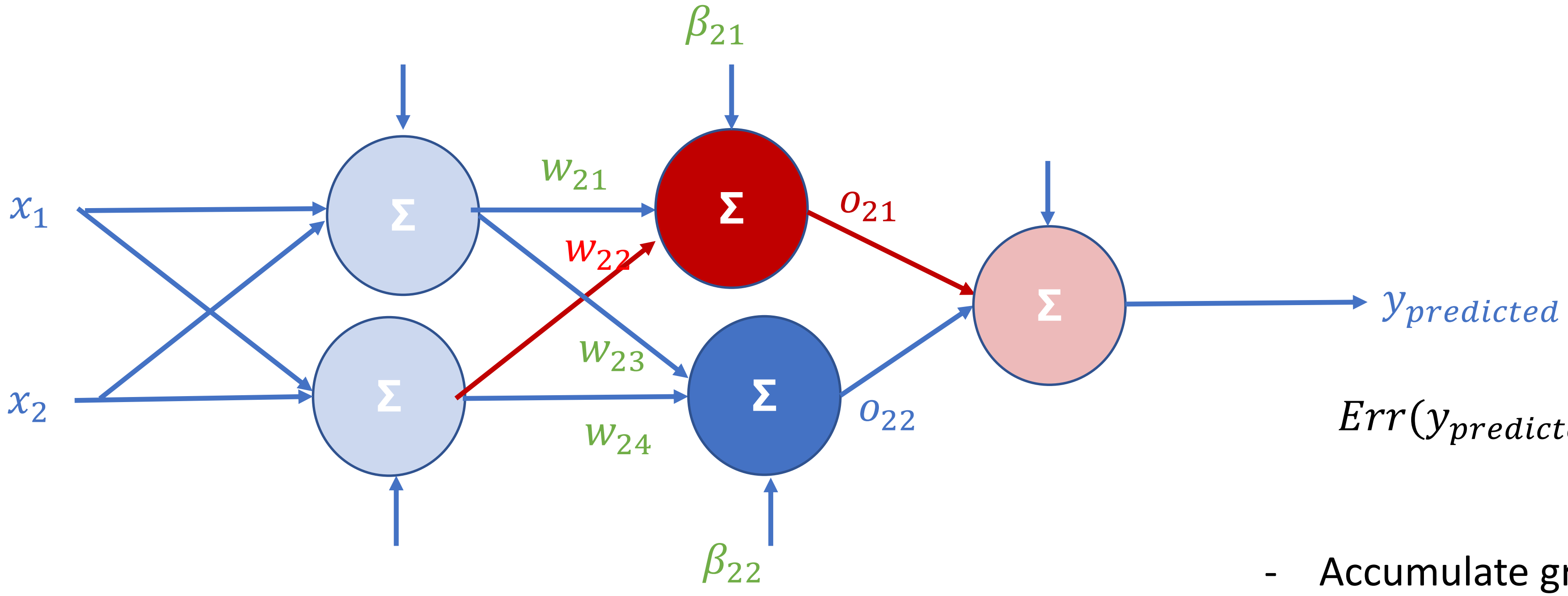
$Err(y_{predicted}, y_{actual})$

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

- 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)



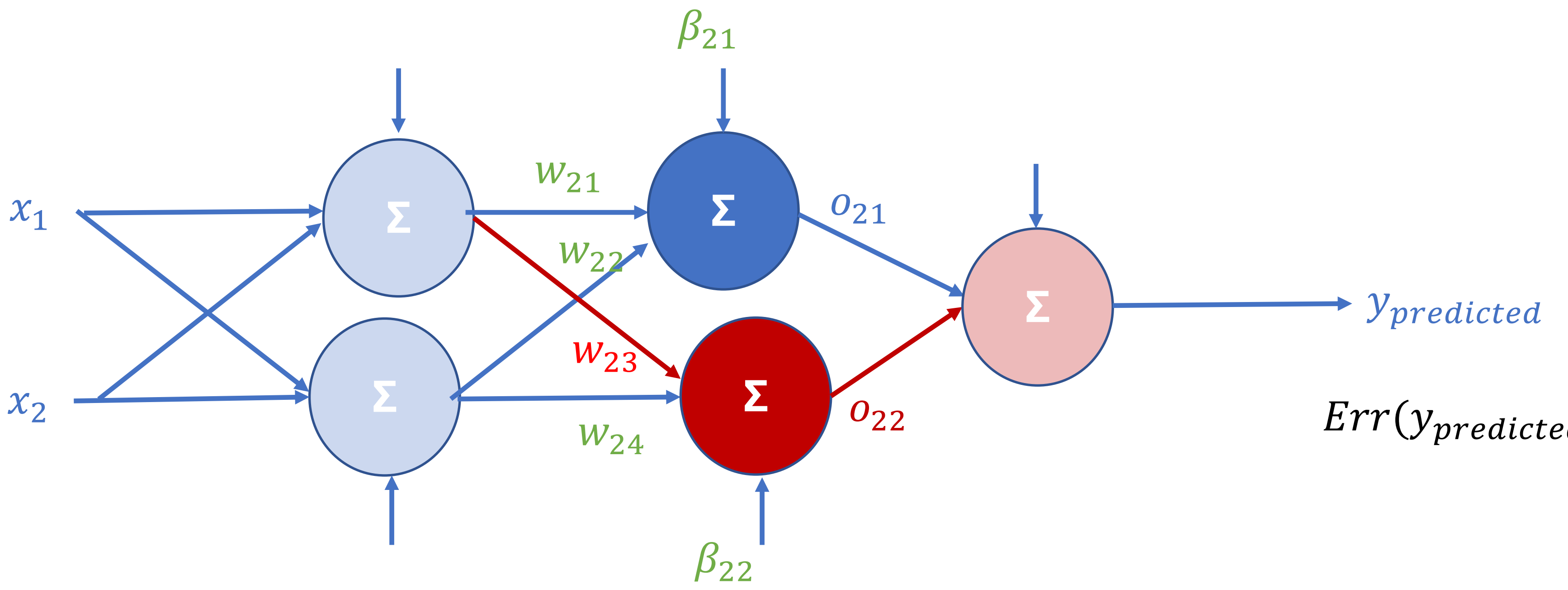
$Err(y_{predicted}, y_{actual})$

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

- 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)

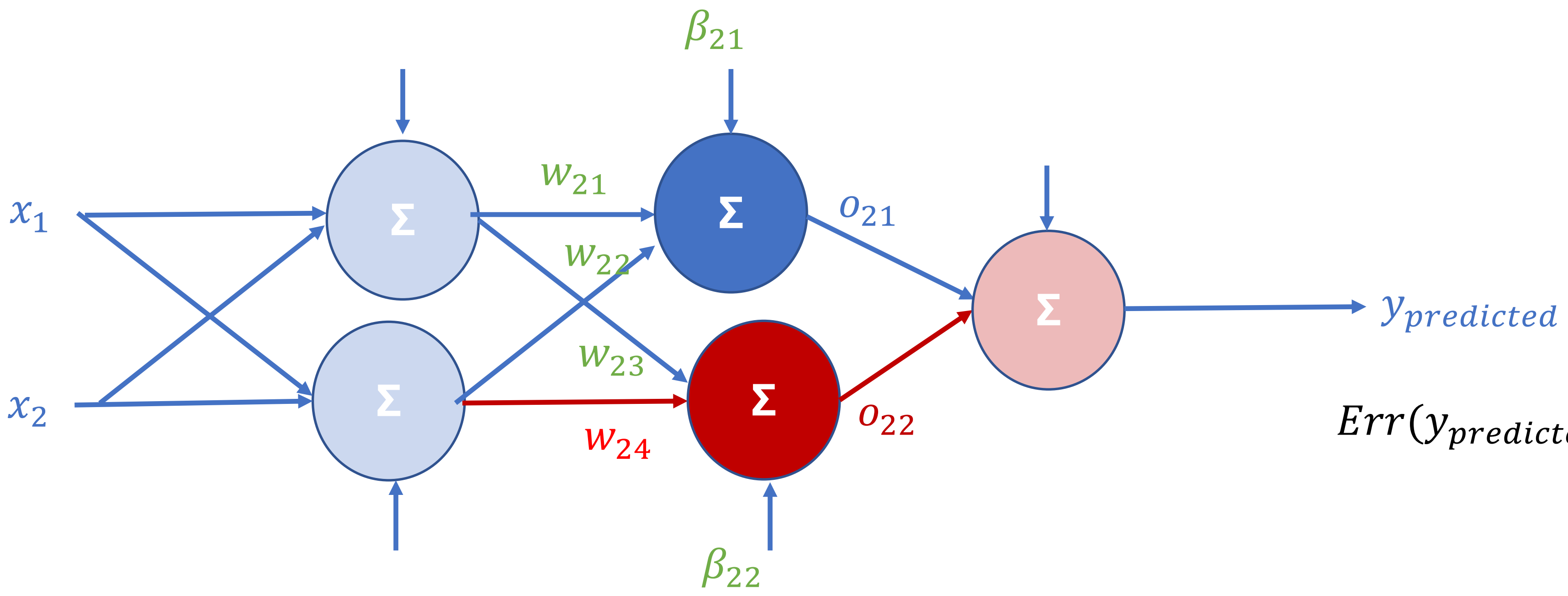


$$Err(y_{predicted}, y_{actual})$$

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

Then

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

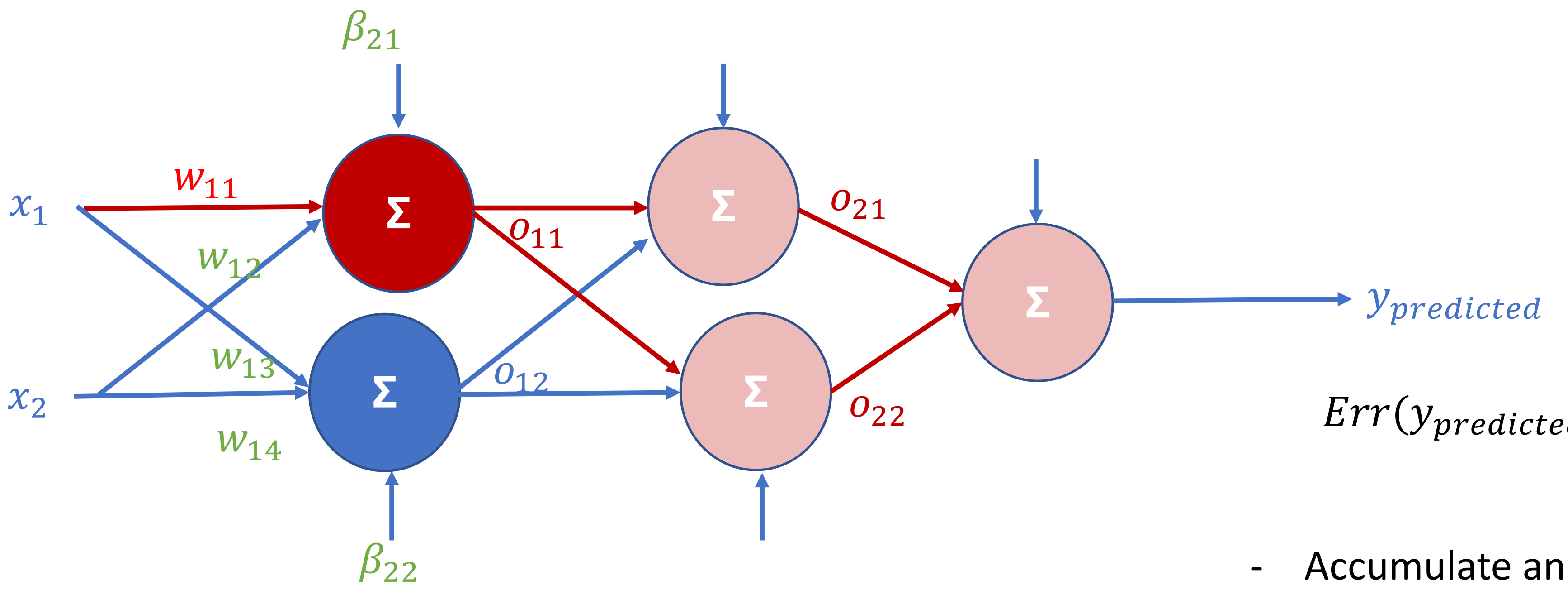


$Err(y_{predicted}, y_{actual})$

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

Then

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



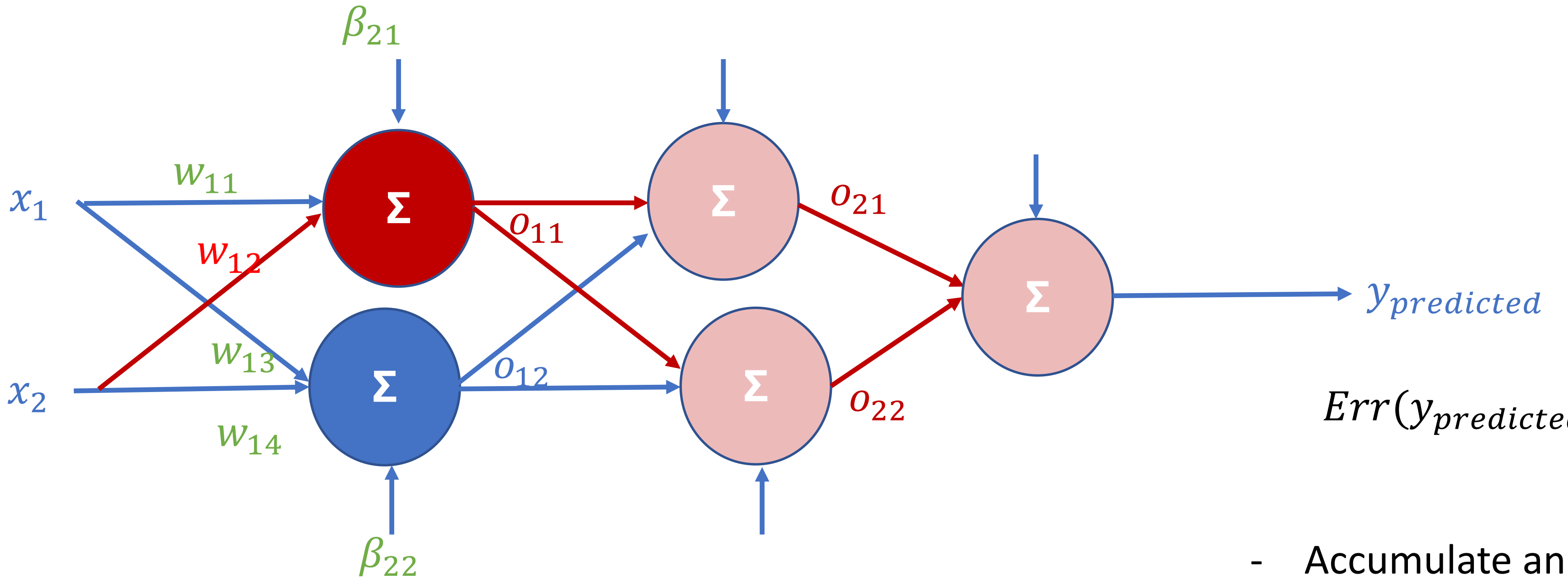
$Err(y_{predicted}, y_{actual})$

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

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

Then

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



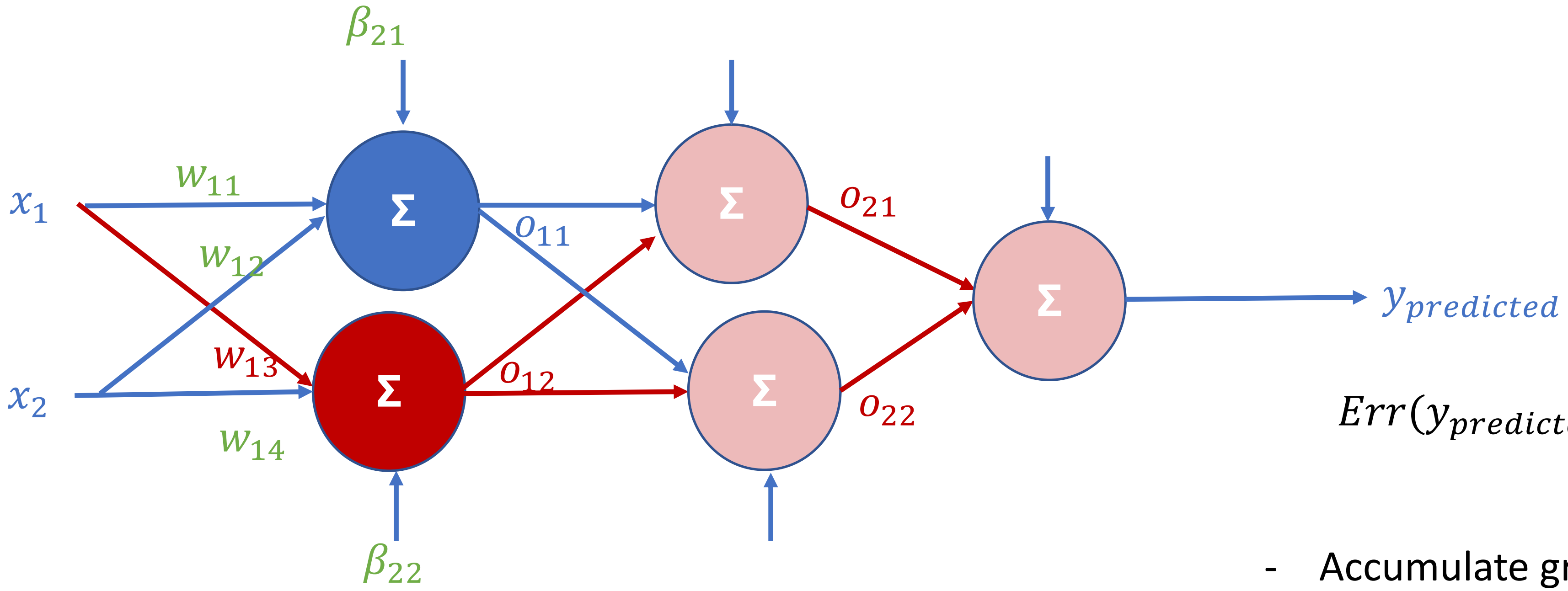
$Err(y_{predicted}, y_{actual})$

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

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

Then

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



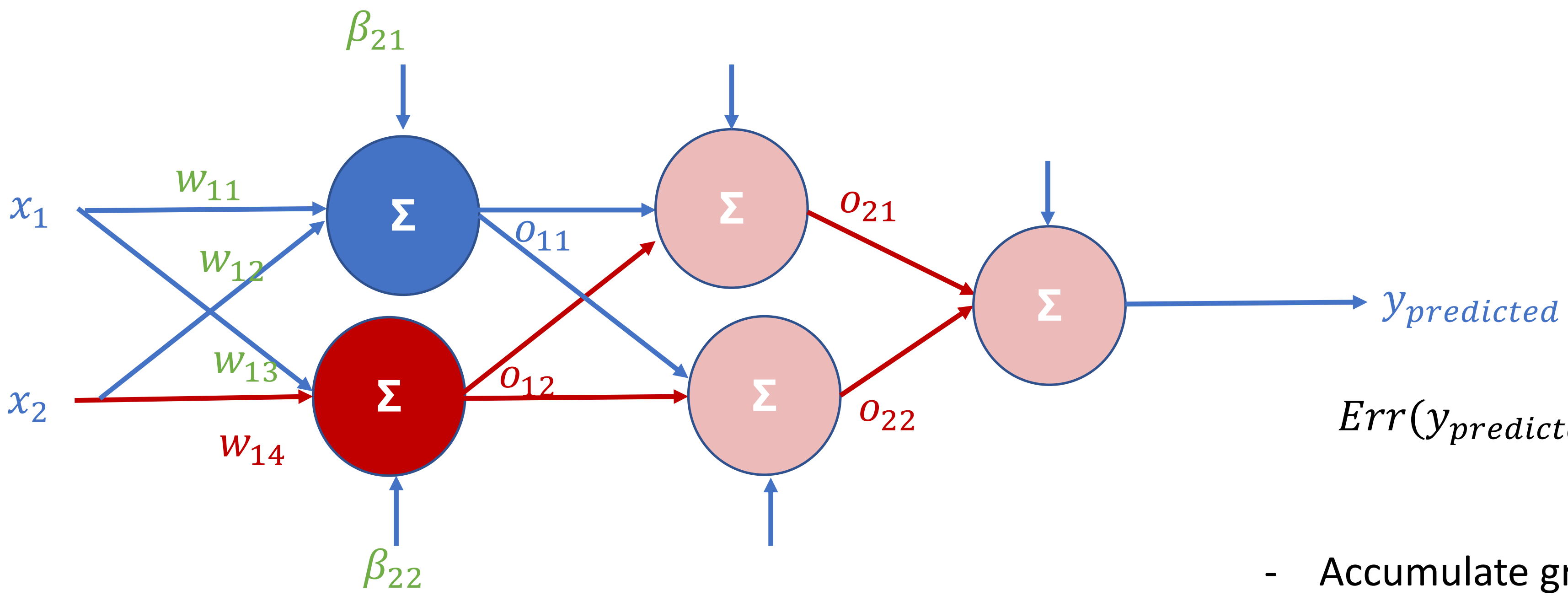
$Err(y_{predicted}, y_{actual})$

- Accumulate gradients of error w.r.t inputs

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

Then

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



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

- Accumulate gradients of error w.r.t inputs

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

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)

In Python

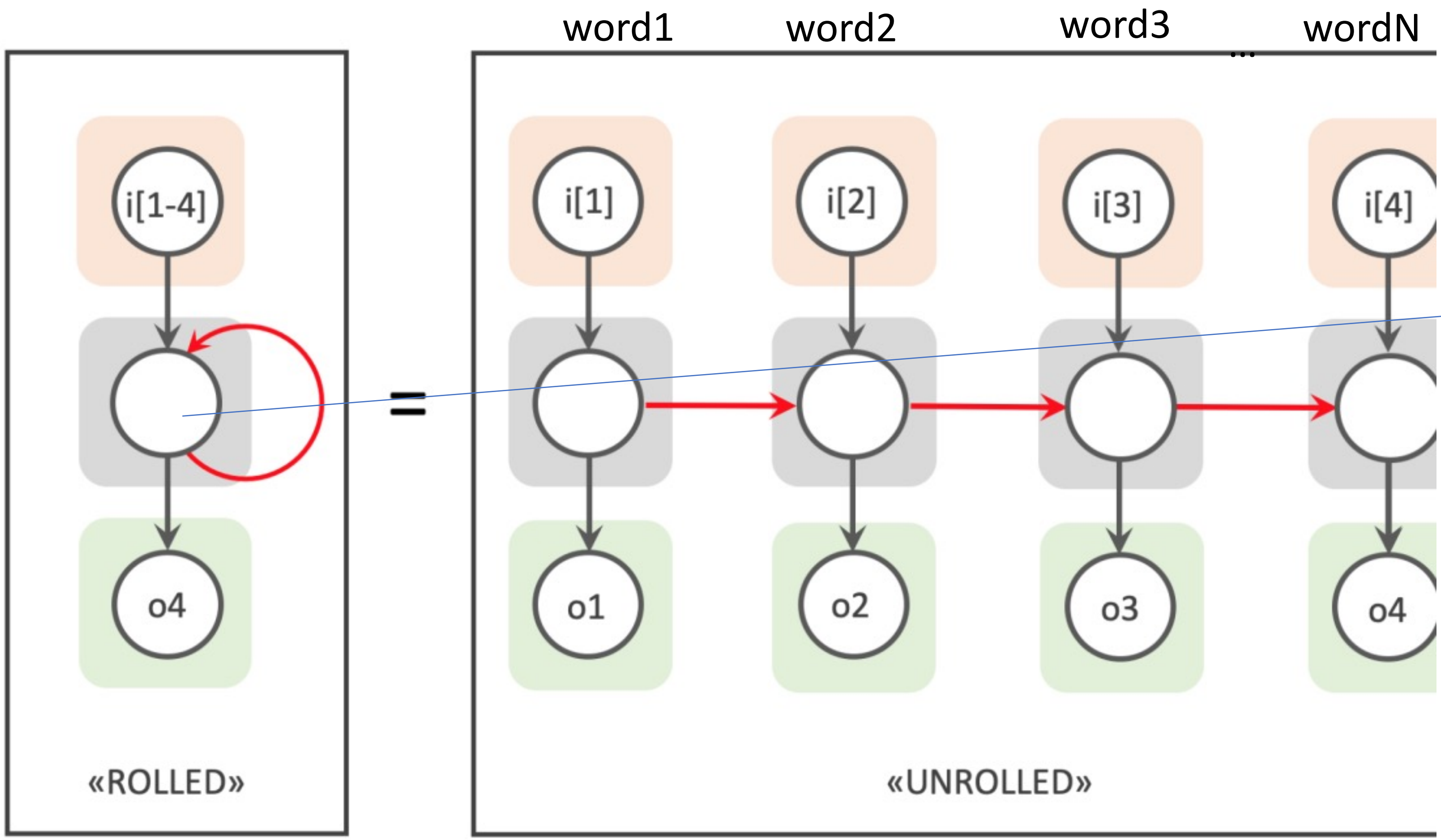
```
model = Sequential()
model.add(Dense(16, input_shape=(vocab_size,), activation='relu')) # 16 units in the hidden layer
model.add(Dense(8))
model.add(Dense(1, activation='sigmoid')) # Output layer with sigmoid activation for binary classification

# Compile the model
model.compile(optimizer='sgd', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

Training:

```
model.fit(X_train.toarray(), train_labels, epochs=50, batch_size=16, verbose=2)
```

Some Advanced Neural Networks

Recurrent Neural Nets

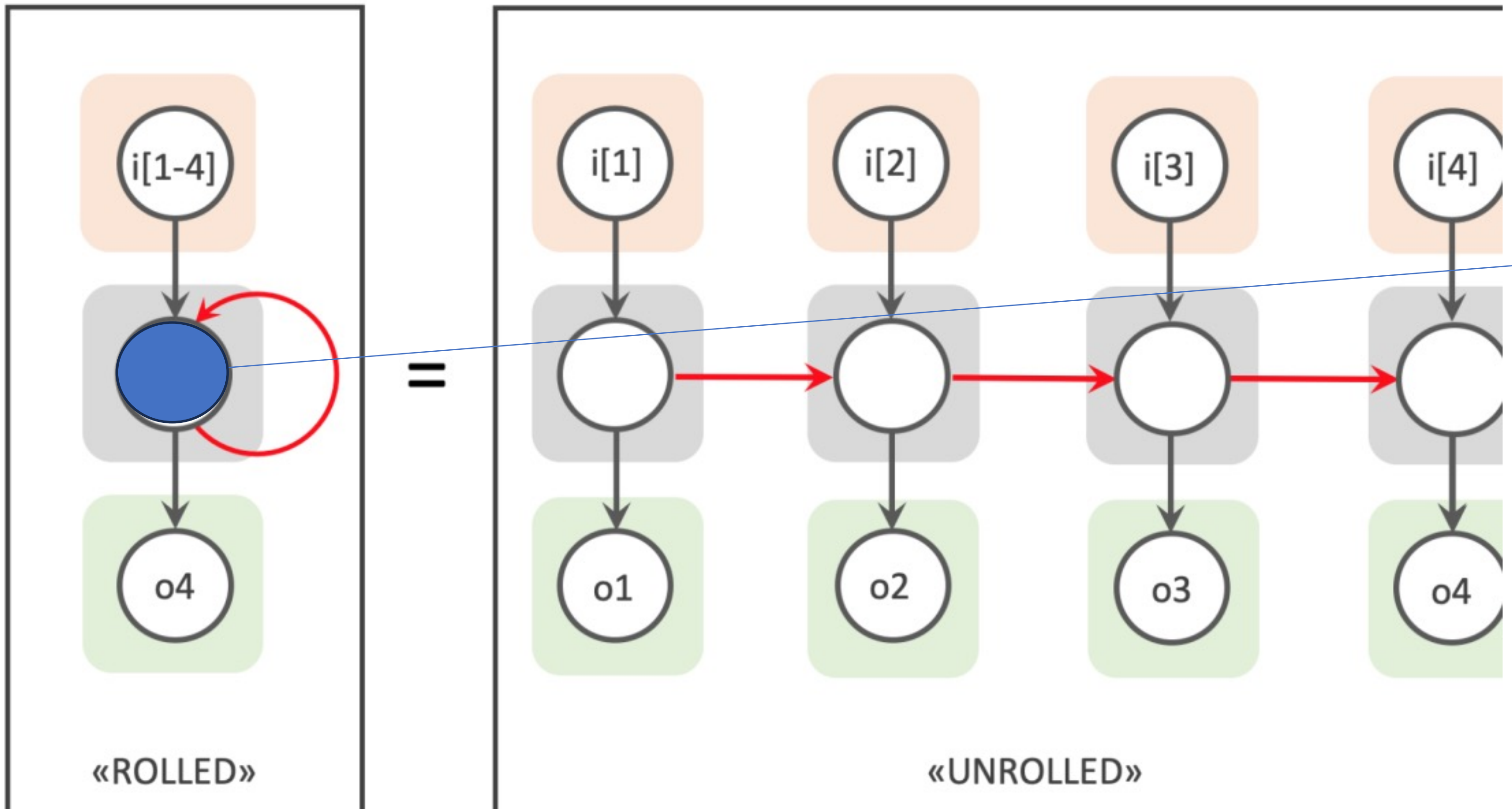


Perceptrons
+ Activations

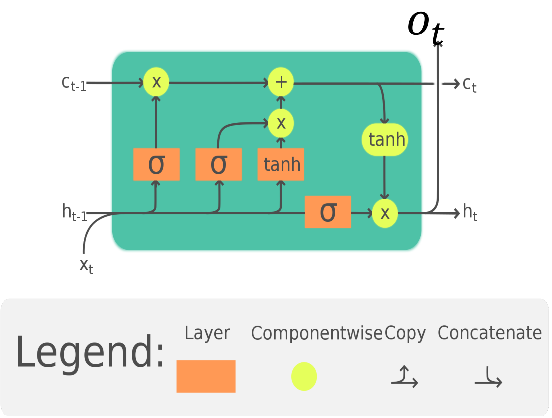
Output (encoded features)

Some Advanced Neural Networks

Long Short Term Memories (Schmidhuber et al, 1997)



LSTM cells



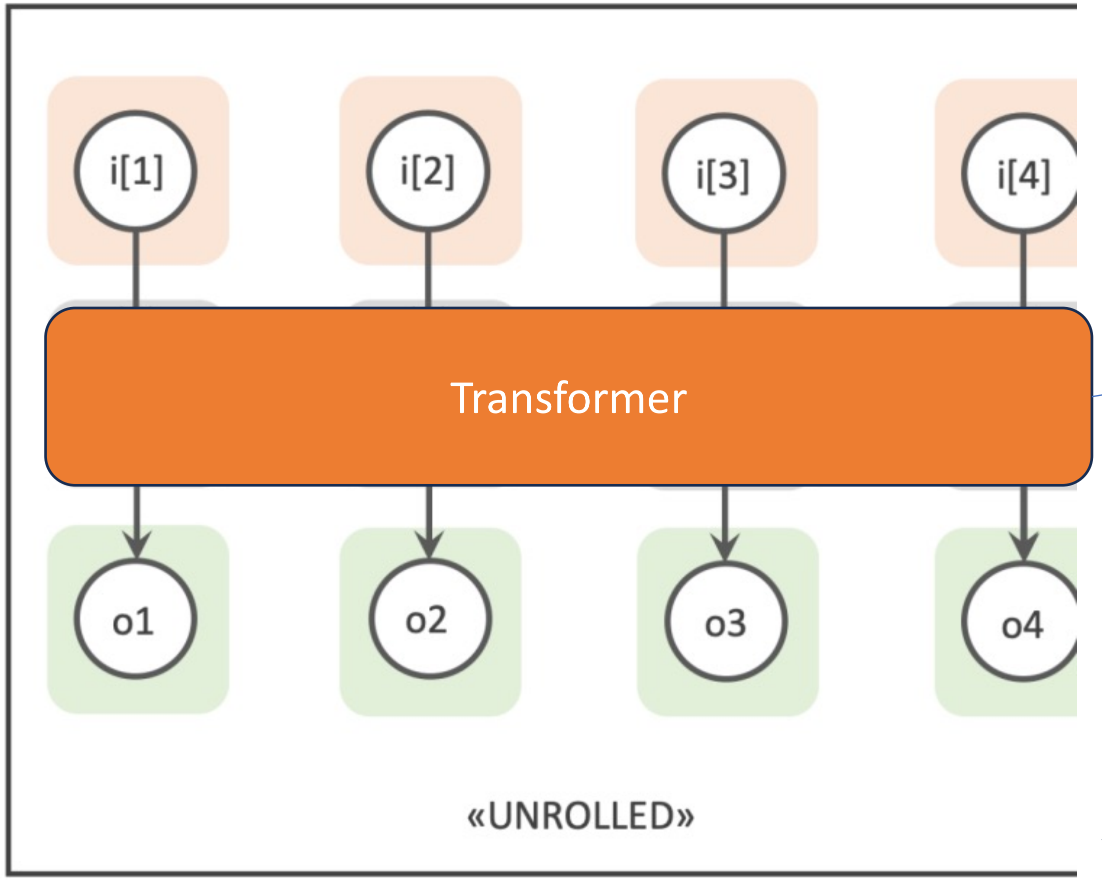
Output (encoded features)

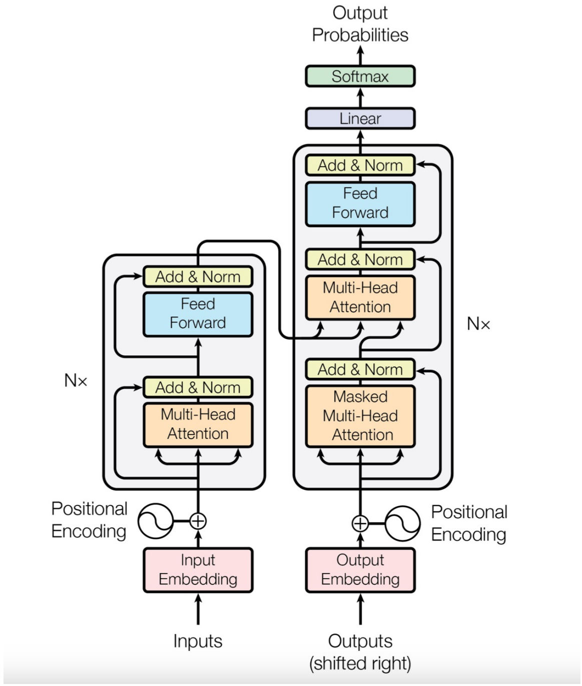
In Python

```
# Define the LSTM model
model = Sequential()
model.add(Embedding(max_features, 128, input_length=maxlen))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
validation_data=(x_test, y_test))
```

Some Advanced Neural Networks

Transformers (Vaswani et al, 2018)

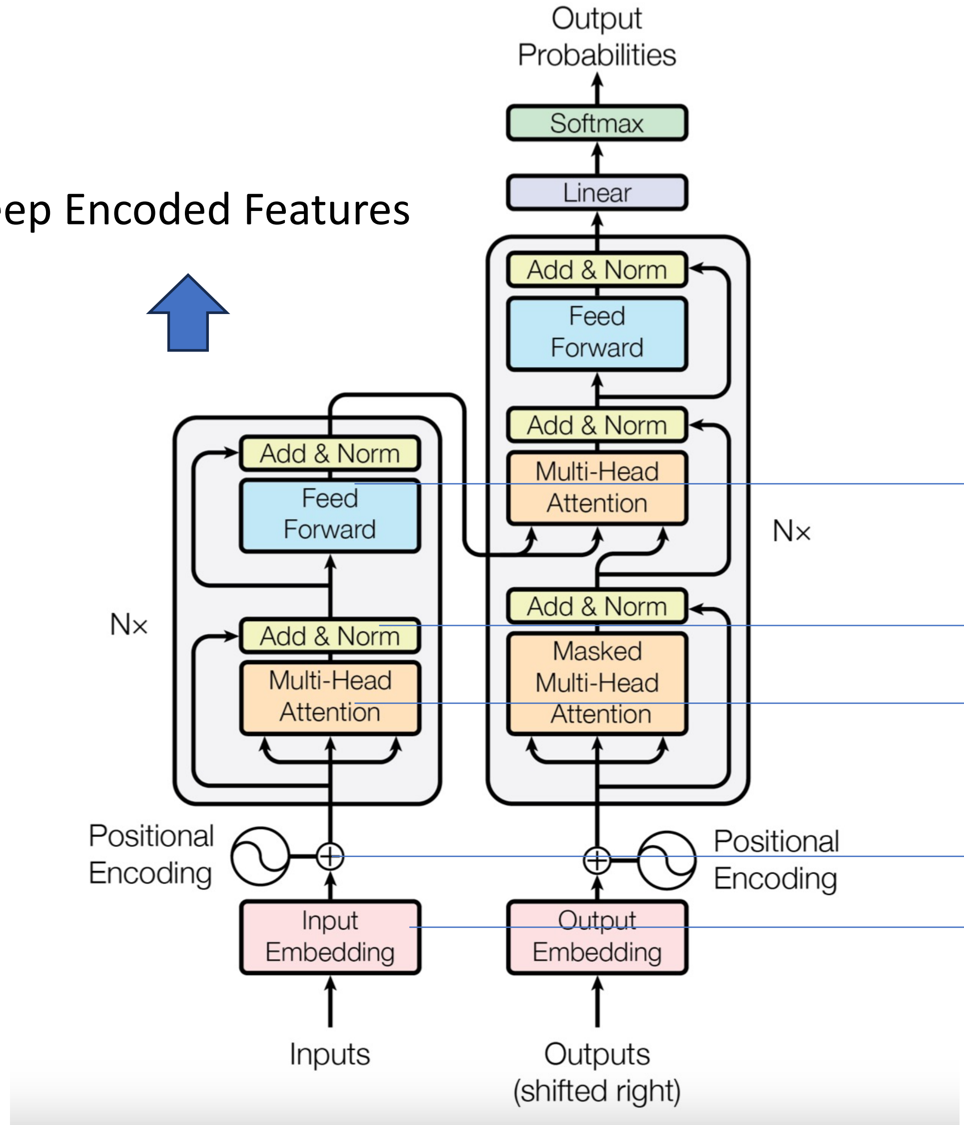




Output (encoded features)

A sneak peek into transformers

Deep Encoded Features



Combining different kinds of concept vectors extracted through self attention and combination of vectors

Addressing vanishing / exploding gradient issues through scaling

Capturing similarities between all pairs of words in the input in multiple dimensions

Dense representations for relative word positions in text

Dense representations for tokens (words)

Tasks solved using These Architectures

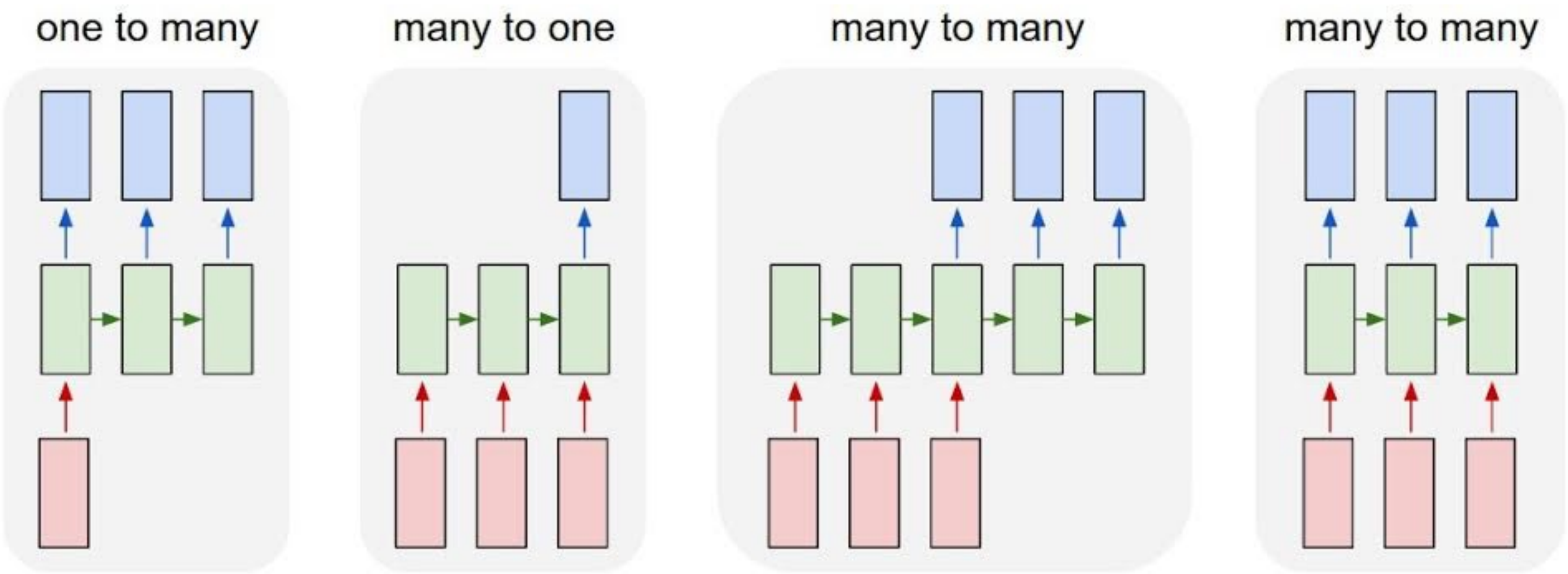


Image Captioning

Video Activity Recog / Text Classification

Video Captioning / Machine Translation

POS Tagging/ Language Modeling

Training Procedure

- Same as Feed Forward Networks: Gradient Back Propagation
- Every path in the network will have gradients accumulated
  - Typically more paths than Feed Forward Networks
- All weights are updated after computing gradients for one batch of data

Popular Python Libraries

Keras

TensorFlow

PyTorch

Language Models and Representation Learning

Neural Networks are not “yet another machine learners”

They do remarkably well in analyzing and understanding variable length inputs and extracting meaningful features (a.k.a represetnations)

How?

How? Representation Learning

- **Feature Engineering from input words?**
  - What do we intend to do?

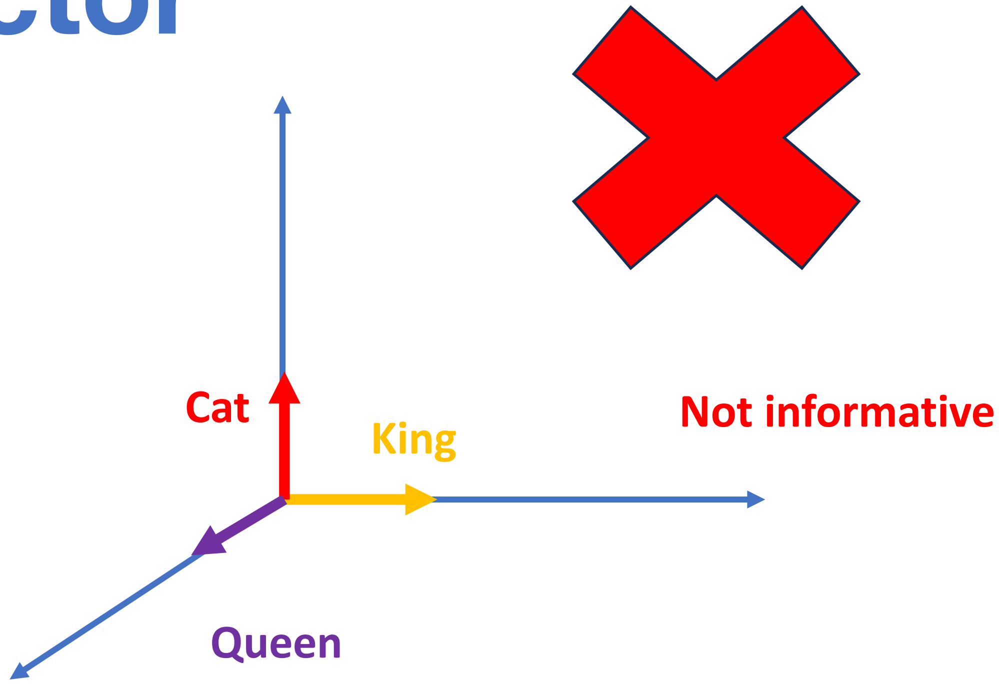
- Extract meaningful representations:
  - In computer understandable numerical forms
  - Should capture relationships between words
    - Synonymy (e.g., “specimen”, “sample”)
    - Antonymy (e.g., ”man”, “woman”)
    - Conceptual similarity (e.g., “Wednesday”, “Monday”) (“USA”,”Canada”)

Traditional representation of a word's meaning

- Dictionary (or PDB), not too useful in computational linguistic research.
- WordNet
  - It is a graph of words, with relationships like “is-a”, synonym sets.
- **Problems:** Depend on human labeling hence missing a lot, hard to automate this process.
- N-hot vectors
  - “Hotel”: [0,0,0,0,0,0,0,0,1,0,0,0,0,0]
  - “Motel”: [0,0,0,0,1,0,0,0,0,0,0,0,0,0]
- **Problems**: Sparse and do not capture deeper relationships
- **TF-IDF** Is slightly better but not good enough

Issues with One-hot vector

Word	1-hot vector
Queen	[1, 0, 0]
King	[0, 1, 0]
Cat	[0, 0, 1]

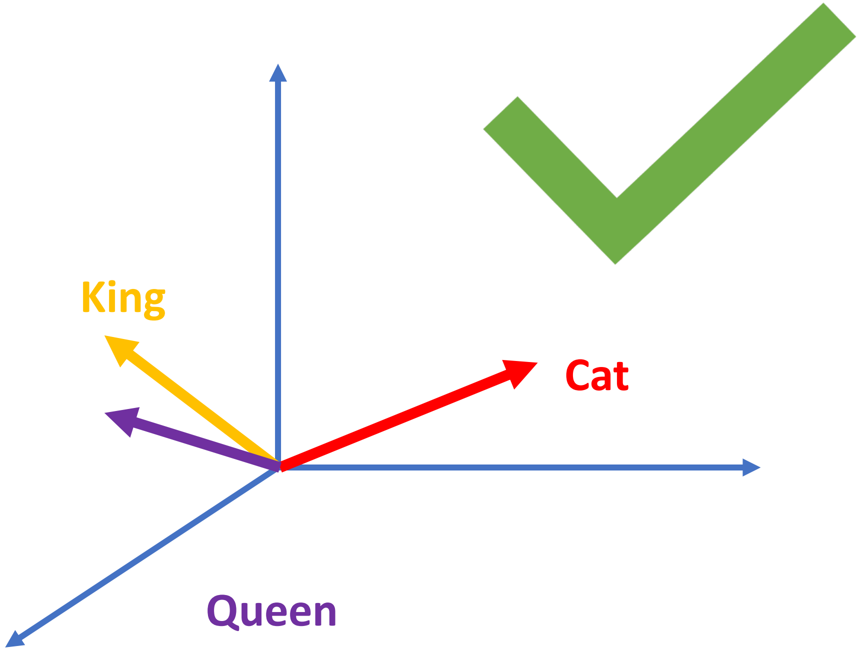


$$D_{cosine}(\text{“}Cat\text{”}, \text{“}King\text{”}) = D_{cosine}(\text{“}Cat\text{”}, \text{“}Queen\text{”}) = D_{cosine}(\text{“}King\text{”}, \text{“}Queen\text{”}) = 1$$

$$D_{Euclid}(\text{“}Cat\text{”}, \text{“}King\text{”}) = D_{Euclid}(\text{“}Cat\text{”}, \text{“}Queen\text{”}) = D_{Euclid}(\text{“}King\text{”}, \text{“}Queen\text{”}) = \sqrt{2}$$

Instead, we need

Word	1-hot vector
Queen	[1.5, -1.3, -0.9]
King	[2.1, -0.7, 0.2]
Cat	[0.3, 1.9, -0.4]



$$D_{cosine}(\text{“}Cat\text{”}, \text{“}King\text{”}) = 1.17, D_{cosine}(\text{“}Cat\text{”}, \text{“}Queen\text{”}) = 1.38,$$
$$D_{cosine}(\text{“}King\text{”}, \text{“}Queen\text{”}) = 0.19$$

$$D_{Euclid}(\text{“}Cat\text{”}, \text{“}King\text{”}) = 3.21, D_{Euclid}(\text{“}Cat\text{”}, \text{“}Queen\text{”}) = 3.45$$
$$D_{Euclid}(\text{“}King\text{”}, \text{“}Queen\text{”}) = 1.38$$

Language Models for Word Embedding

- Language models are statistical or deep learning models that learn to predict the probability of a sequence of words in a sentence or text
- For a sequence of words $W = (w_1, w_2, w_3, ..., w_n)$
- A language model can be expressed as

$$f(X, \theta) \Rightarrow P(W|\theta) = P(w_1|\theta) \cdot P(w_2|w_1, \theta) \cdot P(w_3|w_1, w_2, \theta) \cdot \ldots \cdot P(w_n|w_1, w_2, ..., w_{n-1}, \theta)$$

- Here theta =>model parameters

Different Language Modeling objectives

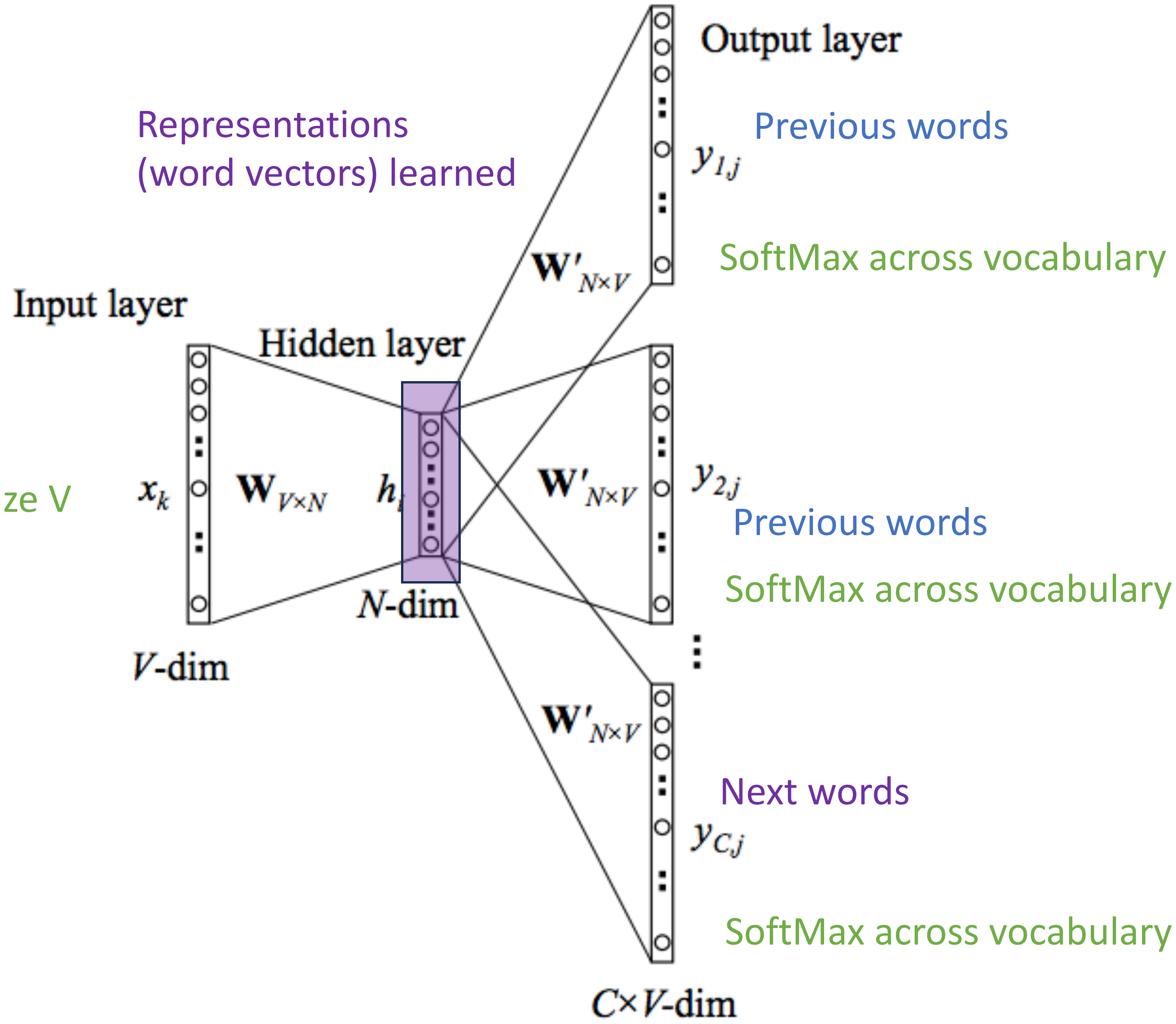
- We can tweak the language modeling objective in different ways for modeling languages
- Some language modeling objectives are
  - **Skip Gram Objective**
  - **Continuous Bag of Words Objective**
  - **Masked Language Model Objective**
  - **Next Sentence Prediction Objective**
  - **Sentence reconstruction from noisy inputs Objective**

Word2Vec: SkipGram

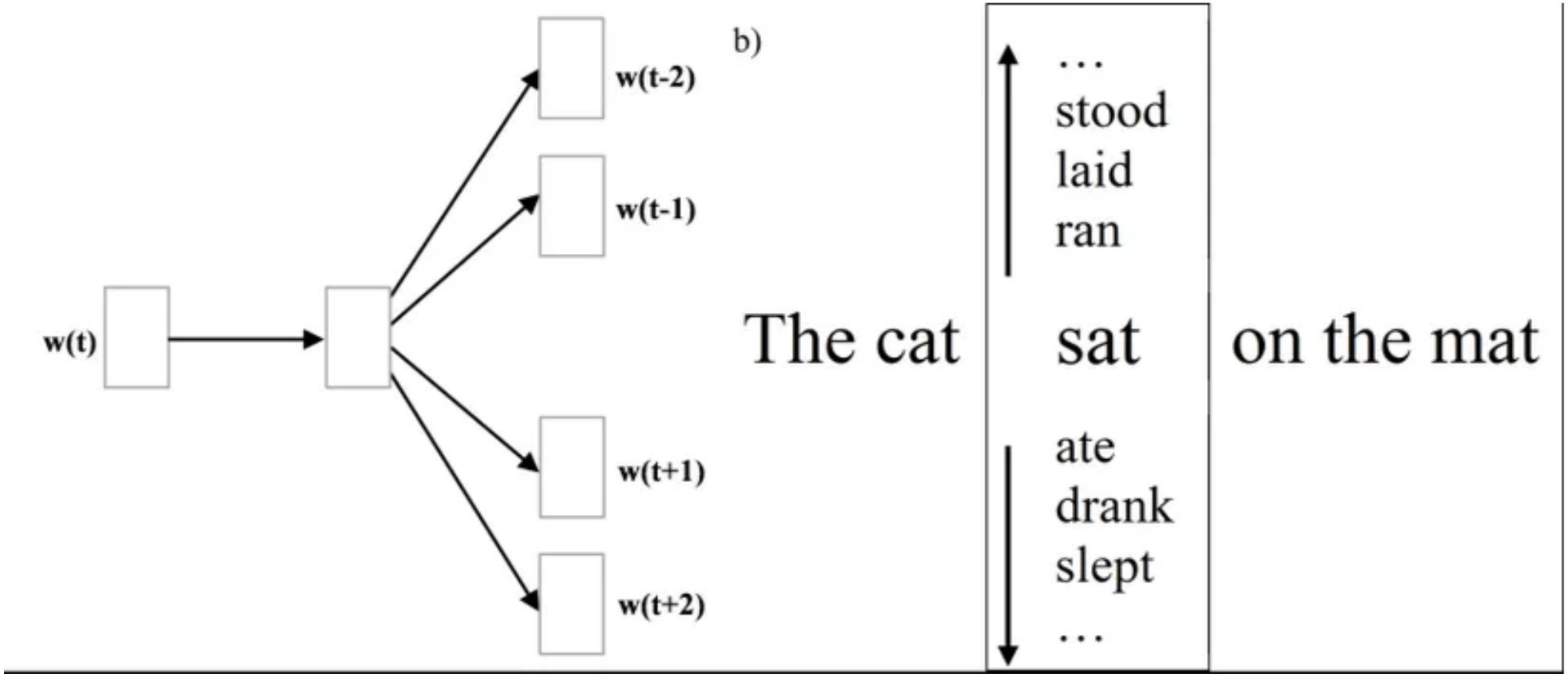
- **Skip Gram Objective :** Given a word can we predict the previous and the next words (or predict surrounding context given an input
- A feed forward network can be designed to perform this task
- **Dataset:**
    - Examples containing <input word, context> can be automatically created using large amount of corpus (e.g., Wikipedia, News Database etc)

SkipGram Model using Feed Forward Nets

Input word (1-hot vector) of size V



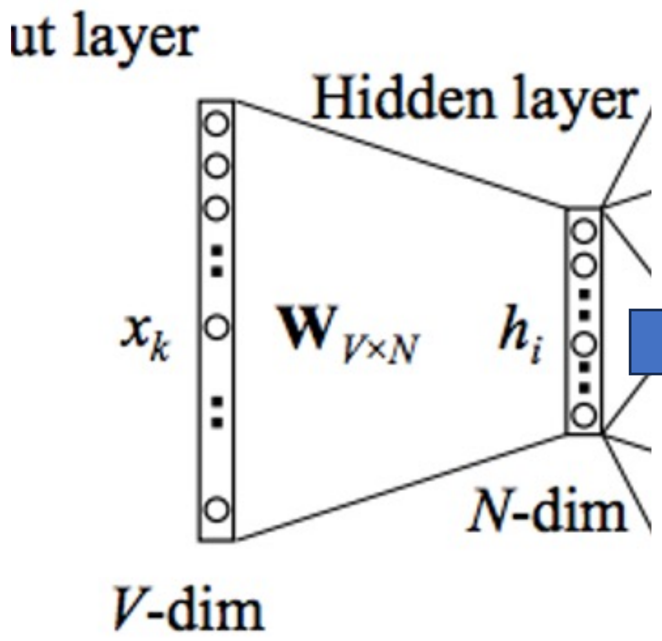
SkipGram Model using Feed Forward Nets



https://towardsdatascience.com/skip-gram-nlp-context-words-prediction-algorithm-5bbf34f84e0c

After Training: During Inference

Word



Feature Vector of size N

These features are the word embeddings we use

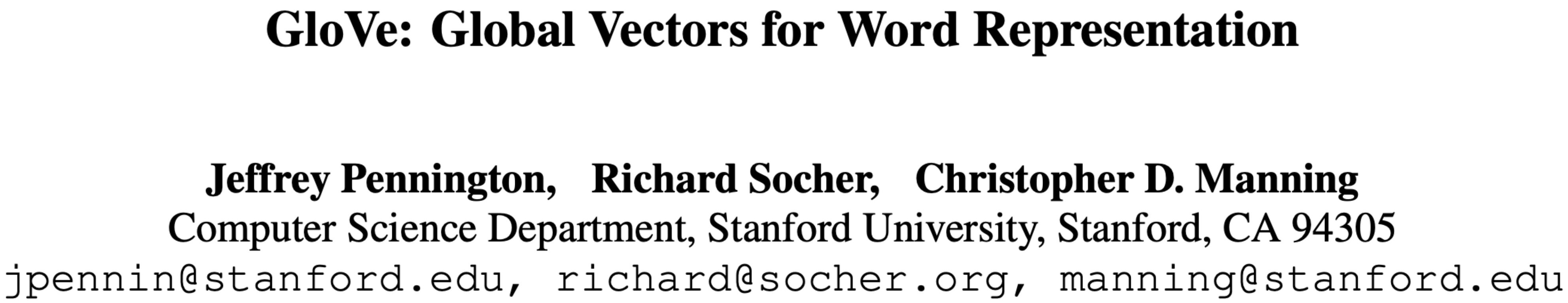
Another Option: The CBOW Model

- What about we predict center word, **given context word, opposite to the skip-gram model**?
- Yes, this is called Continuous Bag Of Words model in the original Word2Vec paper.

Word2Vec: Pros and Cons

- **Pros:**
  - Respects language and order to some extent
  - Efficient Training Process: Simple FFDs
  - Semantic Relationships Preservation
- **Cons:**
  - Loss of local context
  - Does not capture POS variations and word senses
    - Word “bank” will mostly be treated as NOUN
    - Word “bank” will always yield the same vector

GloVE


GloVe: Global Vectors for Word Representation

Jeffrey Pennington, Richard Socher, Christopher D. Manning
Computer Science Department, Stanford University, Stanford, CA 94305
jpennin@stanford.edu, richard@socher.org, manning@stanford.edu

- The GloVe algorithm extracts word vectors by optimizing a defined objective function that captures the statistical co-occurrence information between words

Glove-step-2: Form objective function

$$minimize_{w_1, w_2 \ldots w_V, b_1, b_2, \ldots b_V} \quad (J)$$

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right) \left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

Parameters to be learned

Co-occurrence between word i and j

Co-occurrence between word i and j

$f\left(X_{ij}\right)$ is a weighting function that assigns more weight to less frequent co-occurrences to prevent extremely frequent words from dominating the training.

Glove-step-2: Optimize objective

$$minimize_{w_1,w_2 \ldots w_V, b_1,b_2,\ldots b_V} \ (J)$$

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(\underline{w_i^T \tilde{w}_j} + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

These are the word embeddings and needs to be learned

How?

$$minimize_{w_1,w_2\ \ldots.w_V,b_1,b_2,\ldots b_V}\ (J)$$

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

Initialize randomly and update through gradient descent

How?

$$minimize_{w_1, w_2 \ldots w_V, b_1, b_2, \ldots b_V} \; (J)$$

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

We can randomly initialize a d dimensional vector per word (e.g., d = 50, d = 200)

How to evaluate word embeddings?

- Also by word analogy tasks (Mikolov et al, 2013)
- Solving analogies of the form **"a is to b as c is to ?" or "a:b :: c:"** where you are given three words and you need to find the fourth word that completes the analogy
- Examples:

1. "Man is to woman as king is to ____"
2. "Spain is to Madrid as France is to ____"
3. "Eat is to food as drink is to ____"

Sentence Vectors

Generative Modeling of Text

- Tasks where:
  - Input is a sequence $X = \{x_1, x_2 \dots, x_N\}$ or $\mathbf{X} \in \mathbb{R}^{\boldsymbol{N}}$
  - Output is a sequence $Y = \{y_1, y_2 \dots, y_M\}$ or $\mathbf{Y} \in \mathbb{R}^{\boldsymbol{M}}$

- Example:
  - Text summarization
  - Machine Translation
  - Chat generation

Sequence Generation – Text Summarization Example

SOURCE: *Roger Federer wins a record eighth men's singles title at Wimbledon on Sunday.*

TARGET:

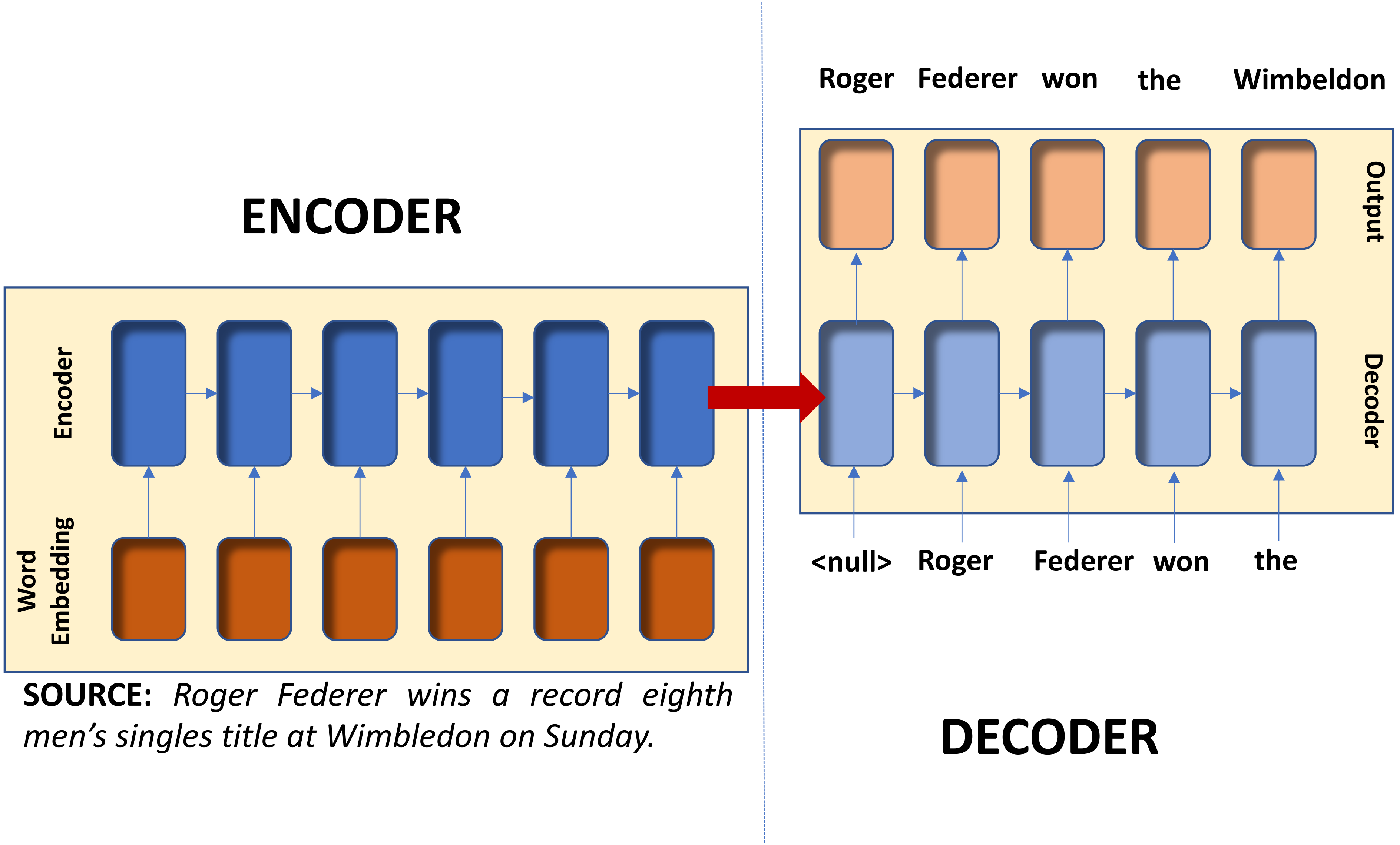
Roger Federer won the Wimbledon

ENCODER

DECODER

REPRESENTATION

Encoder-Decoder Approaches

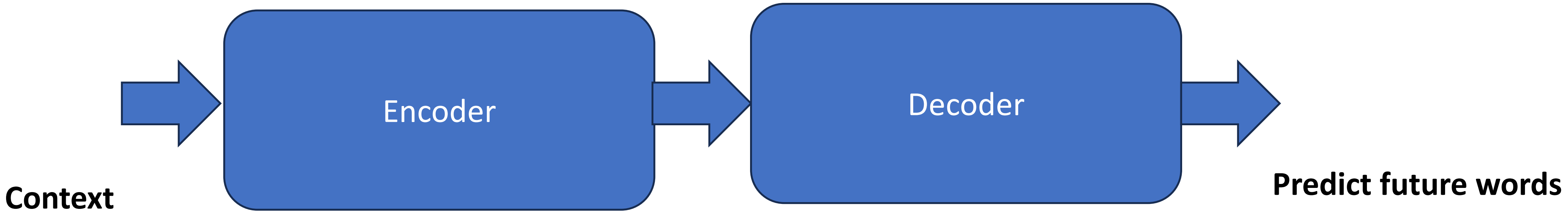


SOURCE: *Roger Federer wins a record eighth men's singles title at Wimbledon on Sunday.*

We have Various choices for the BLUE blocks (RNNs, LSTMs, Transformers)

We can use the same encoder-decoder approach for Language Modeling

- Example



Transformer Based LM: BERT Example

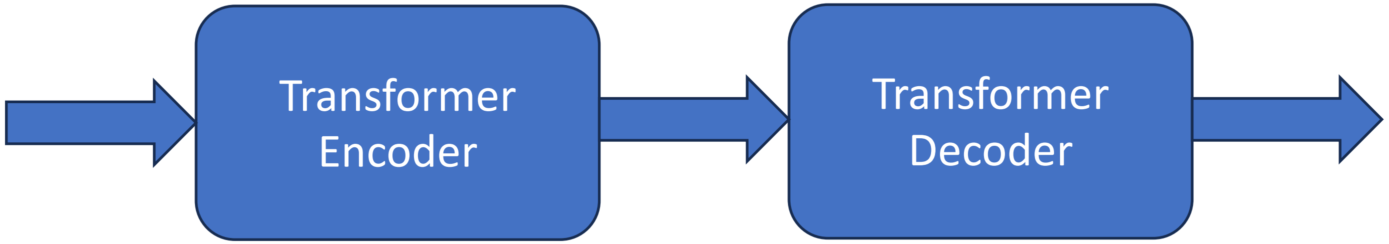
- BERT: **B**idirectional **E**ncoder **R**epresentation **T**ransformers
- Trained with two objectives :
  - Masked token prediction
  - Next sentence prediction

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

Training Task: Masked Token Prediction

[CLS]
Let’s
watch
a
<MASK>
tonight
[SEP]



[CLS]
Let’s
watch
a
movie
tonight
[SEP]

Training Task: Next Sentence Prediction

[CLS]

Let’s

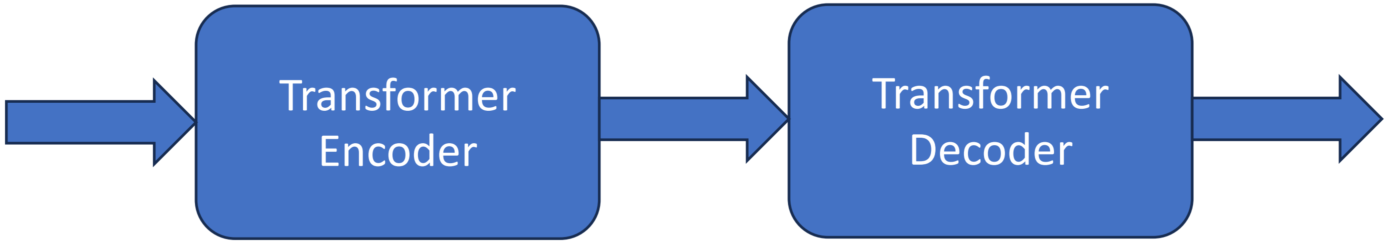
watch

a

movie

tonight

[SEP]



[CLS]

We

will

also

grab

popcorn

[SEP]

Transformer Based LM: BERT Example

[CLS]

Let's

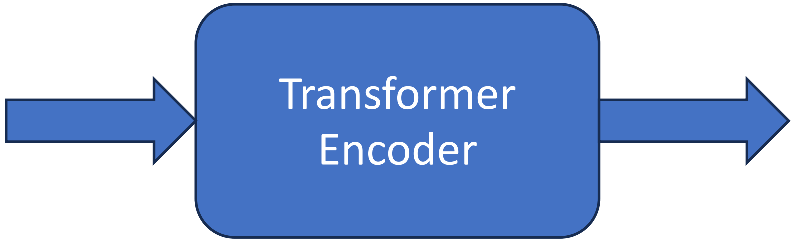
watch

a

movie

tonight

[SEP]



Sentence Embeddings Extraction /
Sentence Feature Extraction

We only use a portion of a network that helps extract features

(also known as encoder)

Sentence Vectors: Pros and Cons

- **Pros:**
  - Contextual understanding
  - Bi-directional learning
- **Cons:**
  - Computational complexity
  - Lack of interpretability
  - Large memory footprint

Next week

- Transfer learning and building NLP applications using transfer learning

Next class

Word and Sentence Embedding explorations

Assignment 5: To be posted today