Unit 4 – Explainability for Text data

What is a Transformer?

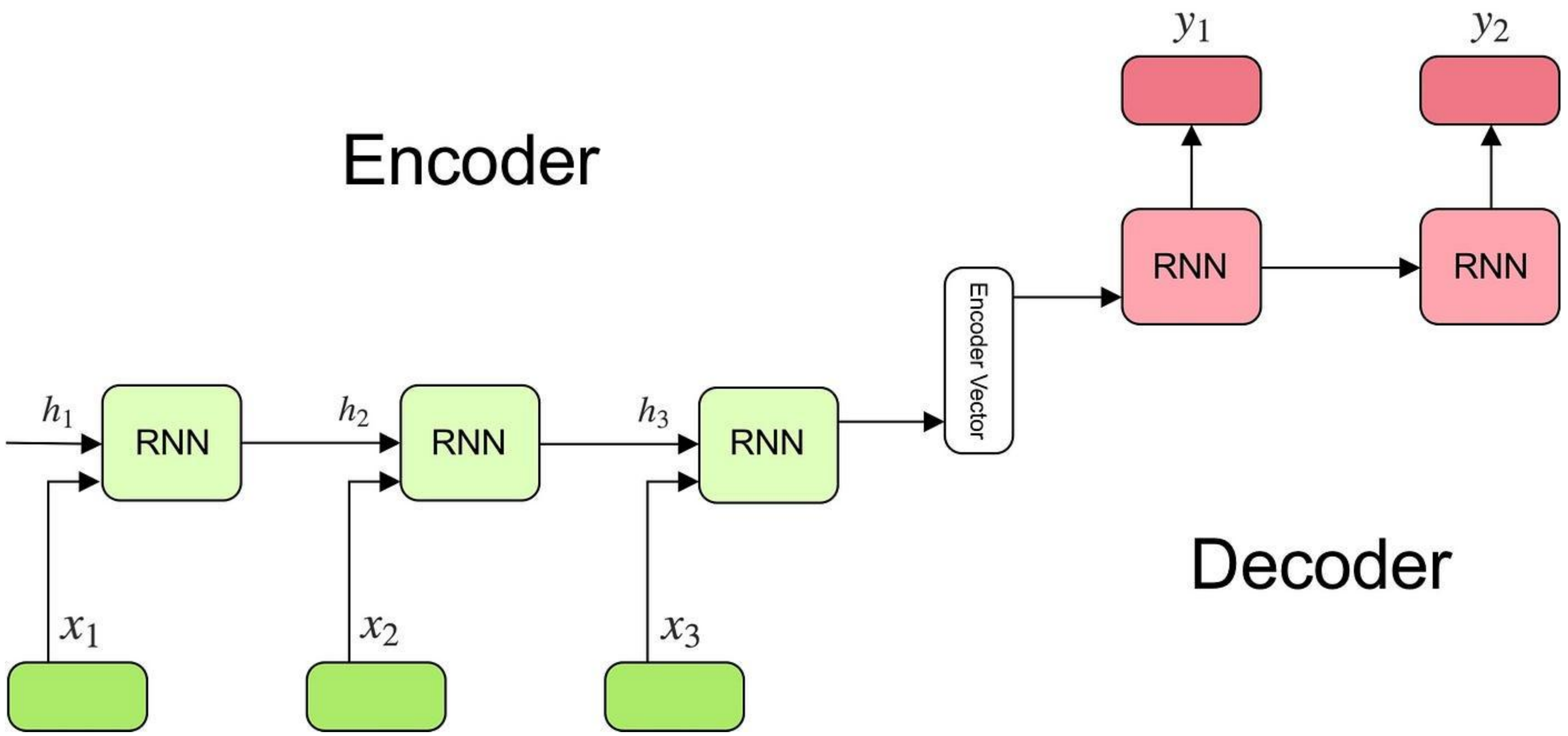
It is a Neural Network, that uses attention mechanism

The attention mechanism is a neural network mechanism that enables a model to focus on the most relevant parts of the input when producing an output.

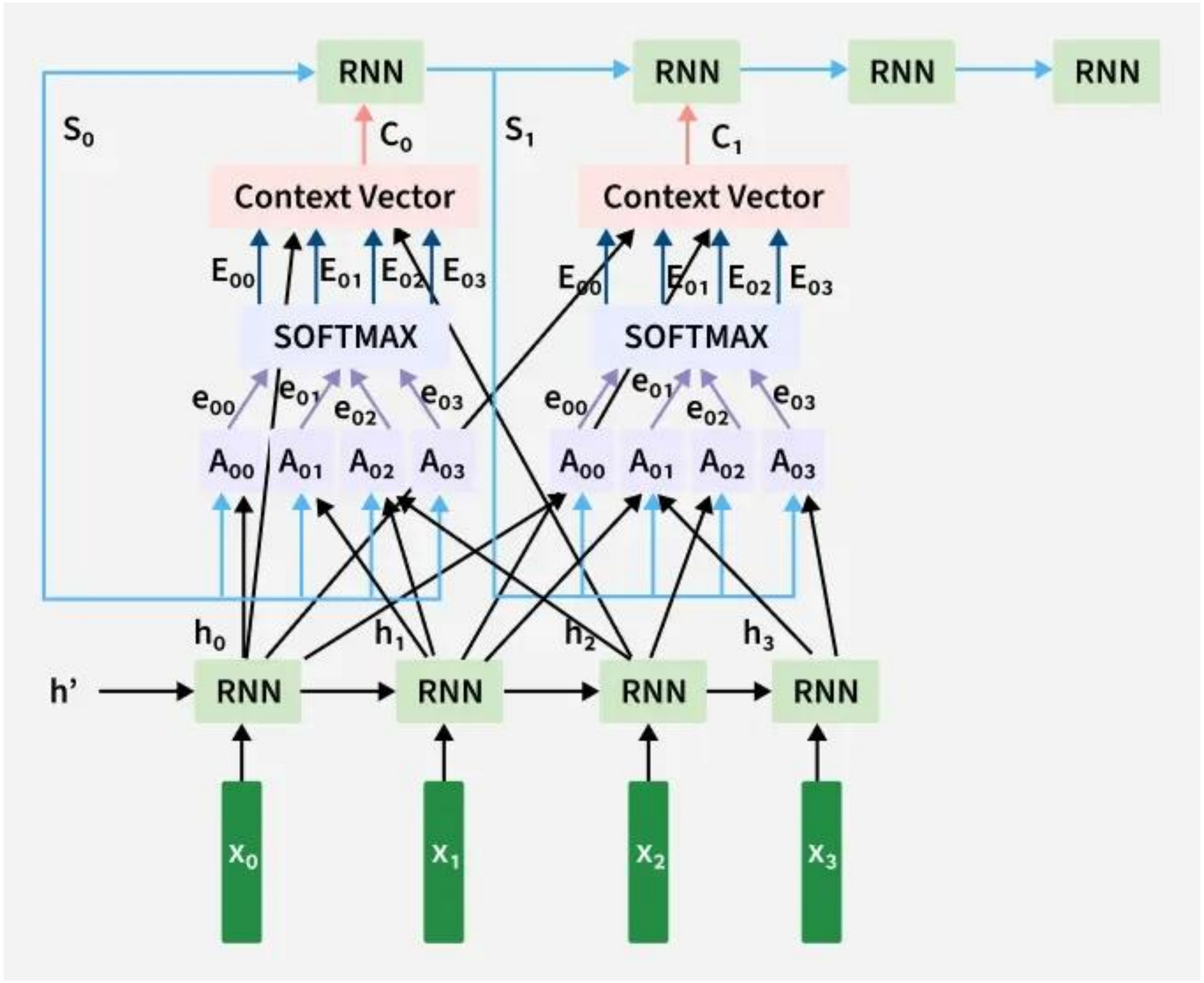
Innovation in attention mechanisms enabled the transformer architecture that yielded the modern large language models (LLMs) that power popular applications like ChatGPT.

Input Processing -> Relevance Scoring -> Weight Assignment -> Weighted Sum -> Output generation

Self attention , Cross attention , Masked Self attention


Encoder
Decoder
h_1
h_2
h_3
RNN
RNN
RNN
x_1
x_2
x_3
Encoder Vector
RNN
RNN
y_1
y_2

Issue	Impact
Memory bottleneck	Hard to compress long sentences into one vector
Loss of important context	Words early in the input are remembered poorly
Weak long-range dependency handling	Meaning often lost in long sequences
Poor translation accuracy on long/complex sentences	Output grammar and semantics degrade


RNN
RNN
RNN
RNN
S₀
C₀
S₁
C₁
Context Vector
Context Vector
E₀₀ E₀₁ E₀₂ E₀₃
E₀₀ E₀₁ E₀₂ E₀₃
SOFTMAX
SOFTMAX
e₀₀ e₀₁ e₀₂ e₀₃
e₀₀ e₀₁ e₀₂ e₀₃
A₀₀ A₀₁ A₀₂ A₀₃
A₀₀ A₀₁ A₀₂ A₀₃
h₀ h₁ h₂ h₃
h'
RNN
RNN
RNN
RNN
X₀
X₁
X₂
X₃

Overview of Building Models with Text

Text to Numeric Quantity (One-hot encoding)

One-hot encoding: Each category is represented as a **binary vector** (all 0s except a single 1 indicating its position).

High dimensionality, Sparsity, Zero Semantic meaning

Tokenization: is the task of splitting up the character sequence into essential pieces, called *tokens*, and discarding irrelevant characters.

Tokens are integer representations of a word or word piece and can be used to split a sentence into words or subwords.

Byte Pair Encoding (BPE)??

To handle rare words, out-of-vocabulary words, Large vocabulary languages

Word Token Embedding

Coverts categorical ID given to a word token into a dense vector

The sequence of token IDs is mapped to dense vectors->**Tokenization** breaks text into individual tokens, while **Embeddings** convert those tokens into numerical vectors that capture semantics and relationships.

Sequence - > Tokens -> Categorical IDs -> Mapped to Embedding Space

The embeddings are a mathematical vector representation that captures different aspects of the data so that similar words are mapped to points that are close to each other.

Semantic similarity becomes geometric closeness

Pretrained embeddings are word (or token) vector representations that are already learned on very large text corpora and can be reused in new tasks instead of training embeddings from scratch.

Static and Contextual Embeddings

How LIME works with Text?

Perturbation of k-dimensional embedding space

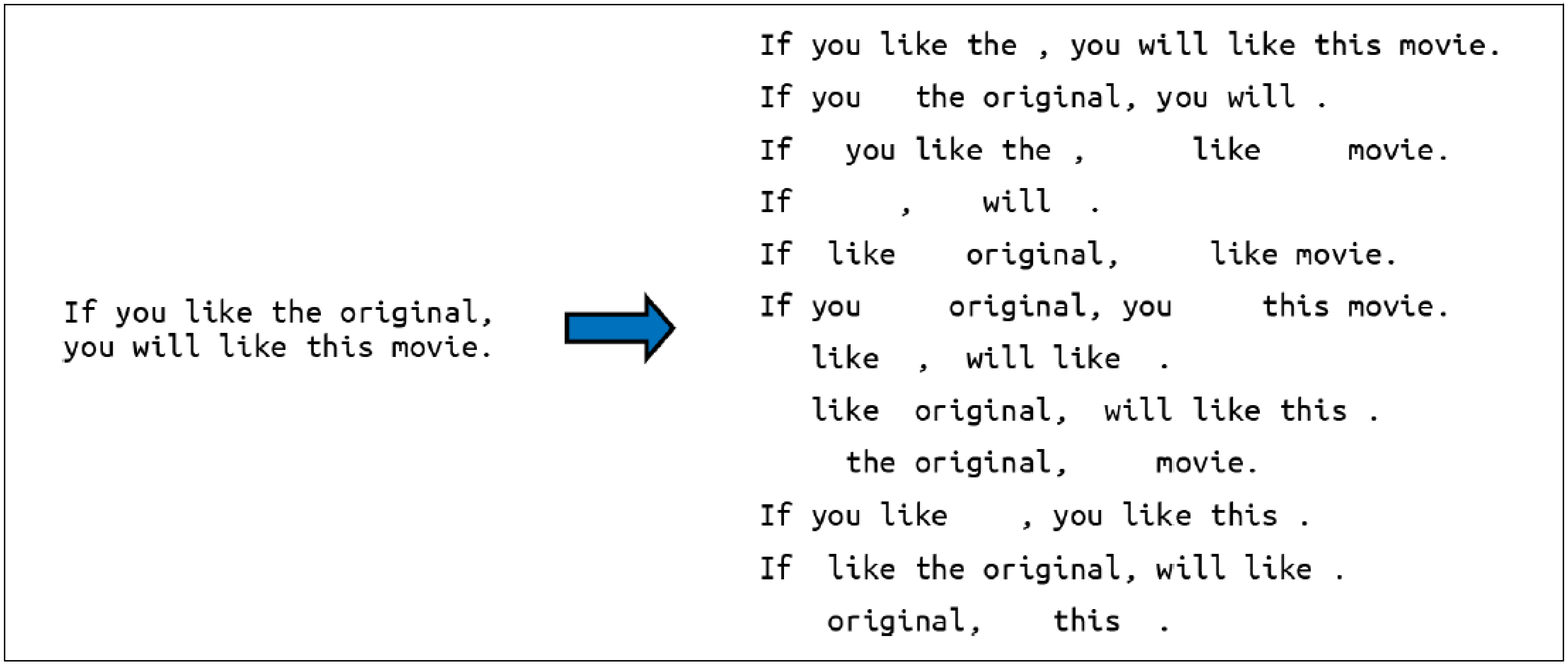
LIME uses an exponential smoothing kernel to define the neighborhood of a data instance -> **higher weights to samples closer to the original instance**

Ensures the explanation is local (focused near the instance being explained) and Makes the surrogate (simple) model approximate the complex model accurately in that neighborhood

LIME is. very sensitive to the kernel width-> Kernel width controls **how large the local neighborhood is** around the instance being explained

Semantic similarity becomes geometric closeness

Individual words are turned “on” or “off ” to create a new, nearby example for inference.



```
If you like the original,
you will like this movie.
```

```
If you like the , you will like this movie.
If you   the original, you will .
If   you like the ,      like     movie.
If      ,    will  .
If  like    original,     like movie.
If you     original, you     this movie.
   like  ,  will like  .
   like  original,  will like this .
     the original,     movie.
If you like    , you like this .
If  like the original, will like .
    original,    this  .
```

$$\text{proximity}_{\kappa}(x_i,x_j) = \sqrt{e^{-d(x_i,x_j)^2/\kappa^2}}$$

Gradient X Input

Simple method for feature attribution

Grad x Input is a saliency method and produces saliency scores proportional to the dot product of the gradient and the input. (Sensitivity and Presence)

Saliency scores for word tokens can be positive or negative, indicating the influence that the token had on the model prediction.

Compute the gradient of the model output with respect to each input feature.

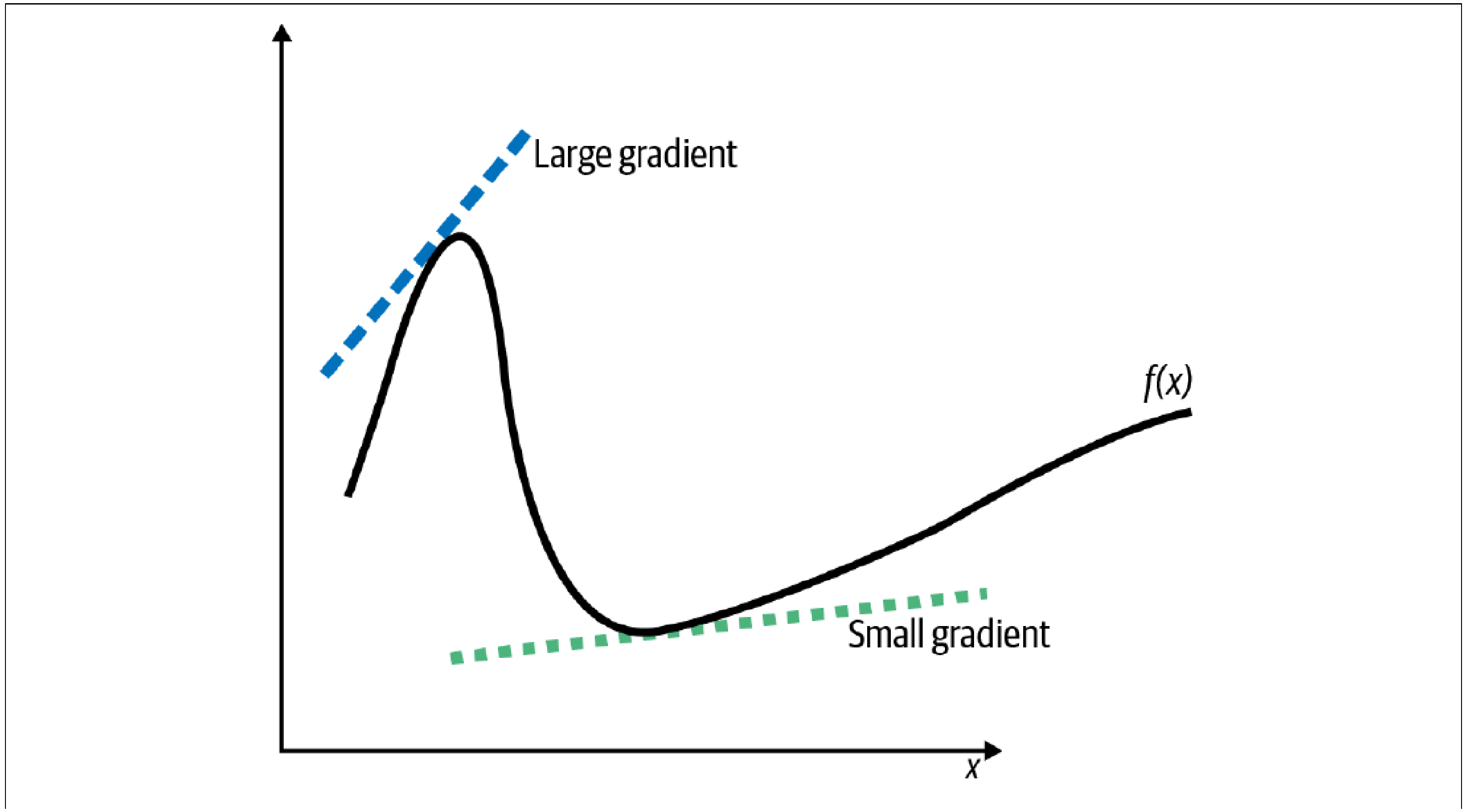
Multiply each gradient by the **actual input value**.

$$\text{Attribution}_i = x_i \times \frac{\partial f(x)}{\partial x_i}$$

It estimates how much changing an input feature affects the prediction, scaled by how strongly that feature is present.

Pros	Cons
• Easy and fast to implement for machine learning libraries like TensorFlow, PyTorch, and JAX; it can be applied to any differentiable model. • Gradient x Input has been shown to be the best performing explainability technique for transformers. • It is available in the Language Interpretability Tool.	• The gradients of a deep neural network can be, and typically are, noisy and sensitive to functions inputs. • Gradient x Input should be used in conjunction with other gradient-based techniques and especially sensitivity methods, like Grad L2-norm (also discussed in this section). • It requires differentiability of the ML model.

Grad L2 Norm



Sensitivity V/S Saliency

Sensitivity answers: *“If I slightly change this input, how much will the output change?”*

Saliency answers: *“Which parts of the input mattered most for this decision?”*

Sensitivity: Measures how pixel-level noise affects the predicted class score.

Saliency: Produces a heatmap showing pixels (e.g., tumor region) that influenced the prediction.

Sensitivity captures the Local responsiveness of the model

Saliency captures the Feature relevance/contribution

Sensitivity-based XAI: ????

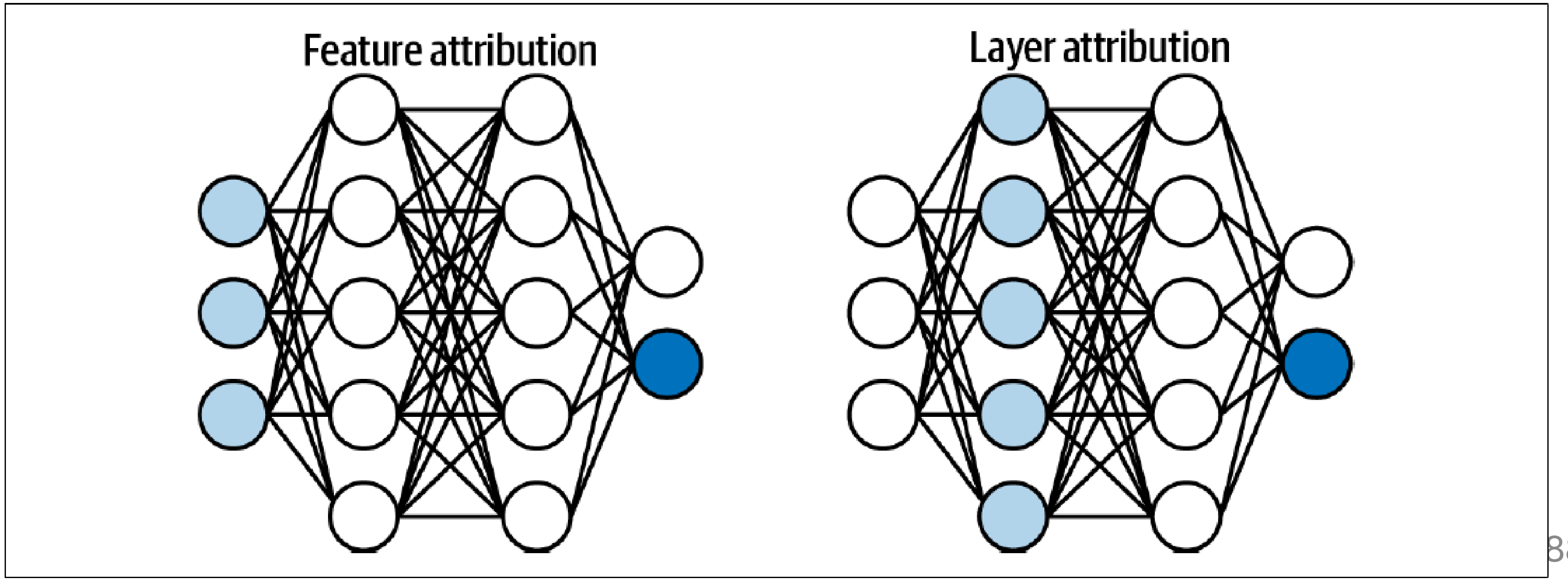
Saliency-based XAI: ????

Layer Integrated Gradients (LIG)

Focuses on a single layer of the network instead of input features;

LIG extends IG by:

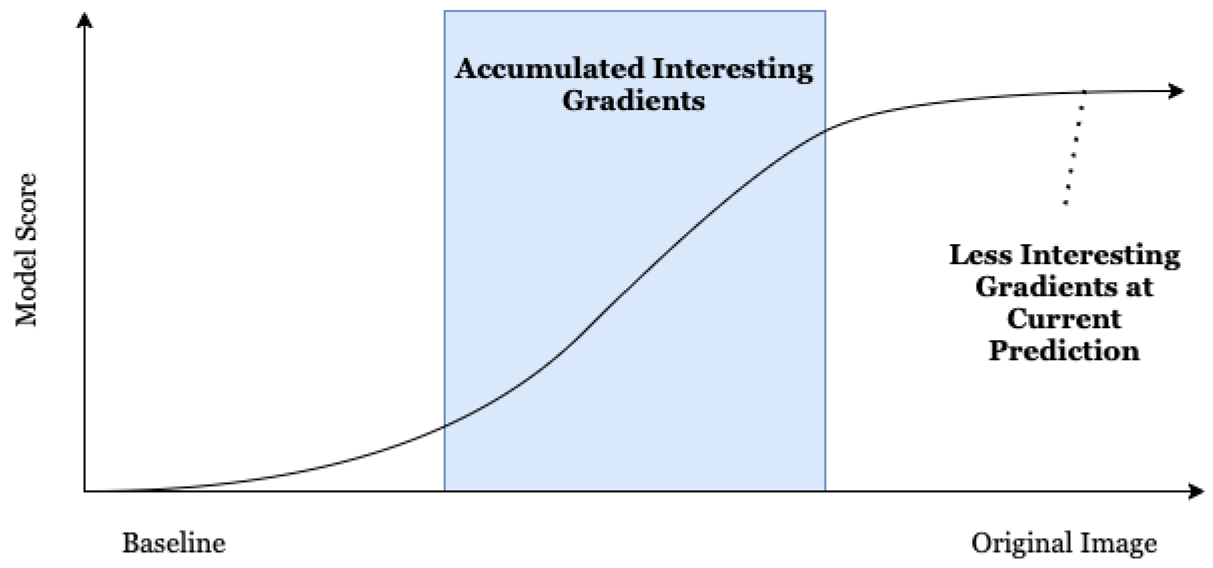
- Selecting an internal layer (usually the embedding layer),
- Treating the embedding vectors as the “effective input,”
- Computing Integrated Gradients with respect to that layer.



Pros

- LIG is useful for text to isolate the embedding layer for computing Integrated Gradients.
- It can be applied to any differentiable model for any data type, images, text, tabular, etc.
- There is an easy and intuitive implementation that even novice practitioners can apply.

- LIG requires differentiability of the model and access to the gradients, so it does not apply well to tree-based models.
- The results can be sensitive to hyperparameters or choice of baseline.


Accumulated Interesting Gradients
Less Interesting Gradients at Current Prediction
Model Score
Baseline
Original Image