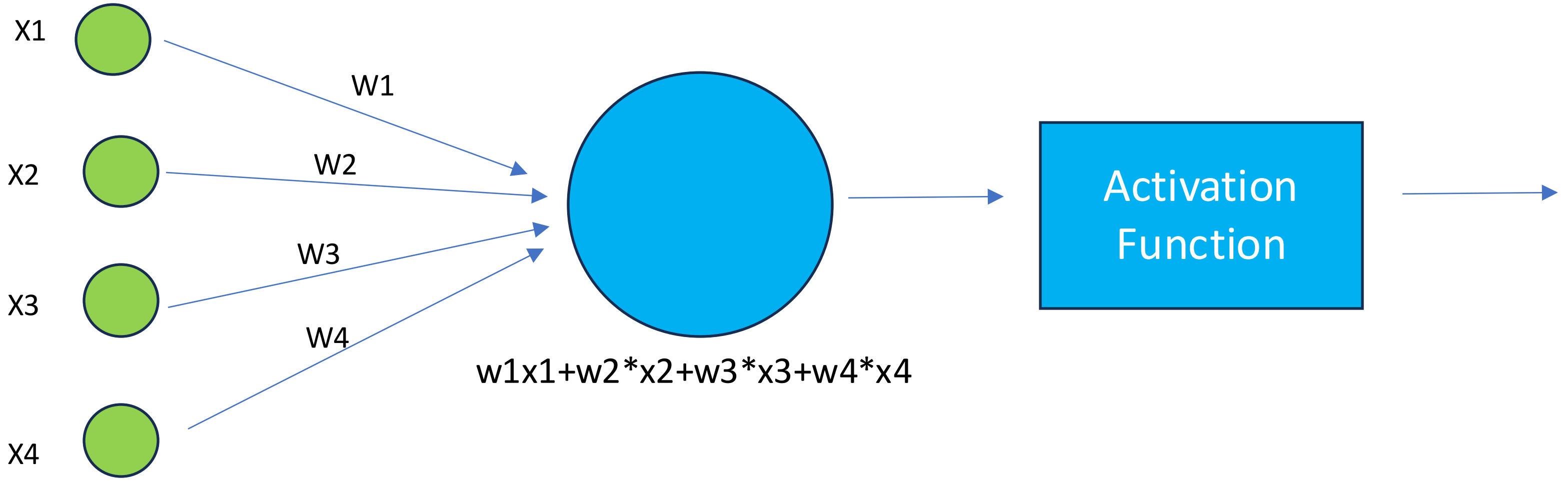
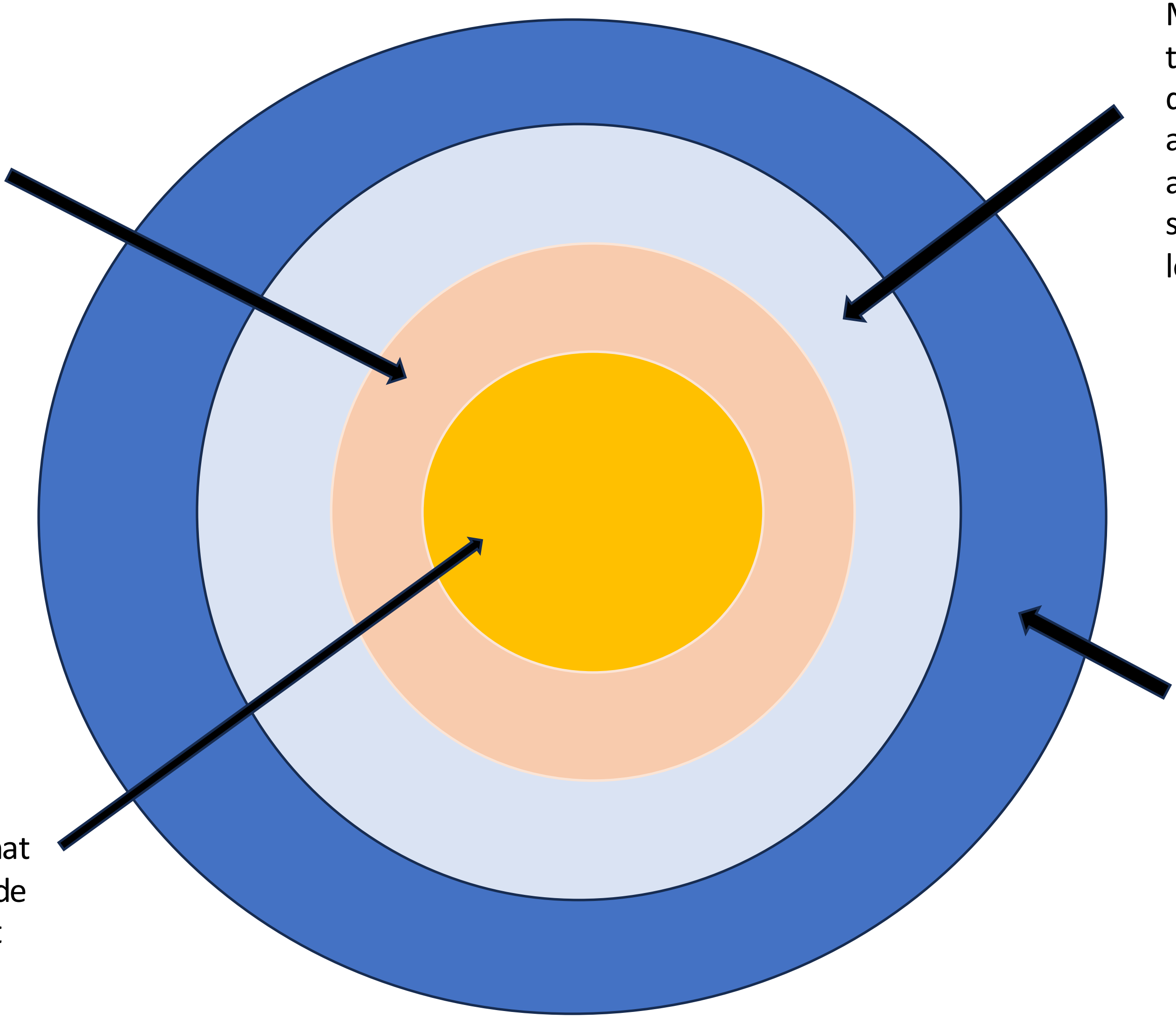

X1
X2
X3
X4
W1
W2
W3
W4
w1x1+w2*x2+w3*x3+w4*x4
Activation Function

Machine Learning (ML)

Machine Learning is a subset of AI that uses advanced algorithms to detect patterns in large datasets, allowing machines to learn and adapt. ML algorithms use either supervised or unsupervised learning methods.

Deep Learning (DL)

Deep Learning is a subset of ML that uses neural networks for in-depth data processing and analytical tasks. DL leverages multiple layers of artificial neural networks to extract high-level features from raw input data, simulating the way human brains perceive and understand the world.

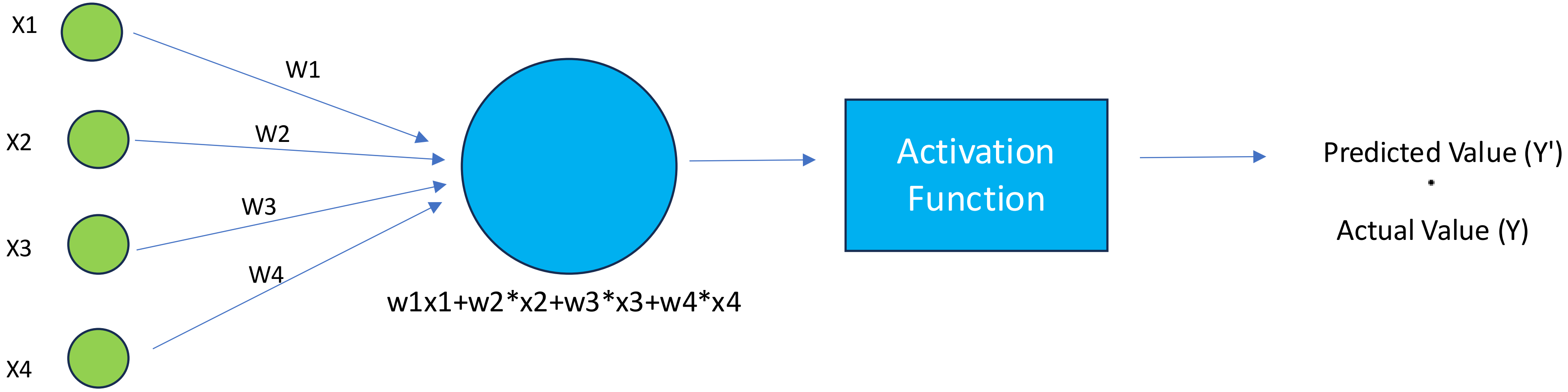


Artificial Intelligence (AI)

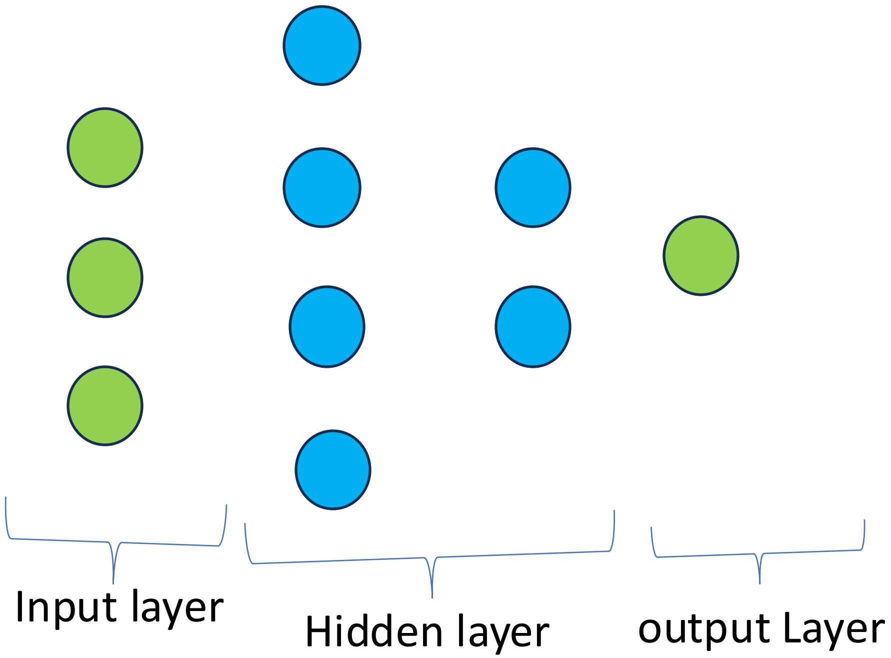
AI involves techniques that equip computers to emulate human behavior, enabling them to learn, make decisions, recognize patterns, and solve complex problems in a manner akin to human intelligence.

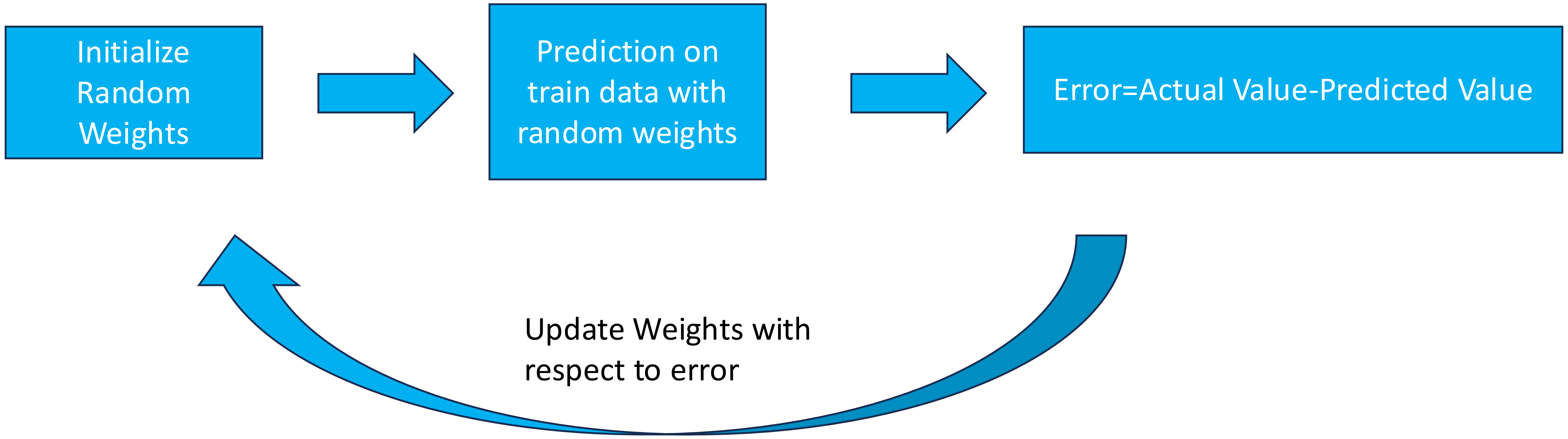
Generative AI

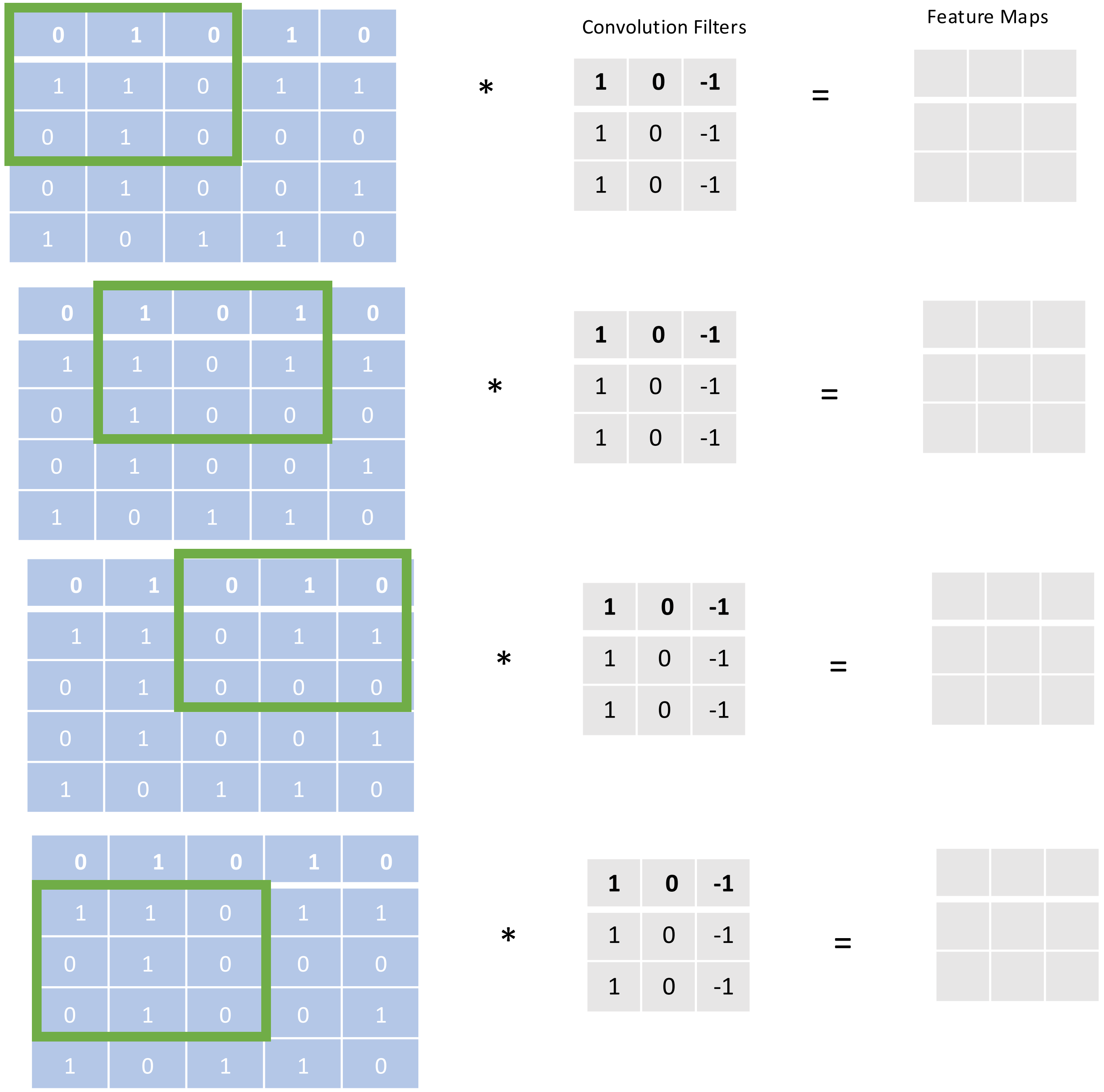
Generative AI is a subset of DL models that generate content like text, images, or code based on provided input. Trained on vast datasets, these models detect patterns and create outputs without explicit instruction, using a mix of supervised and unsupervised learning.

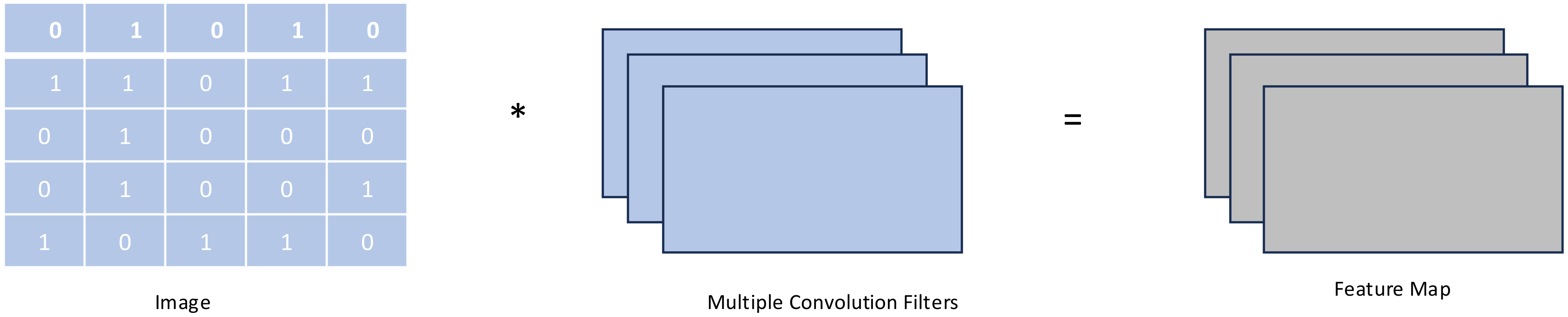


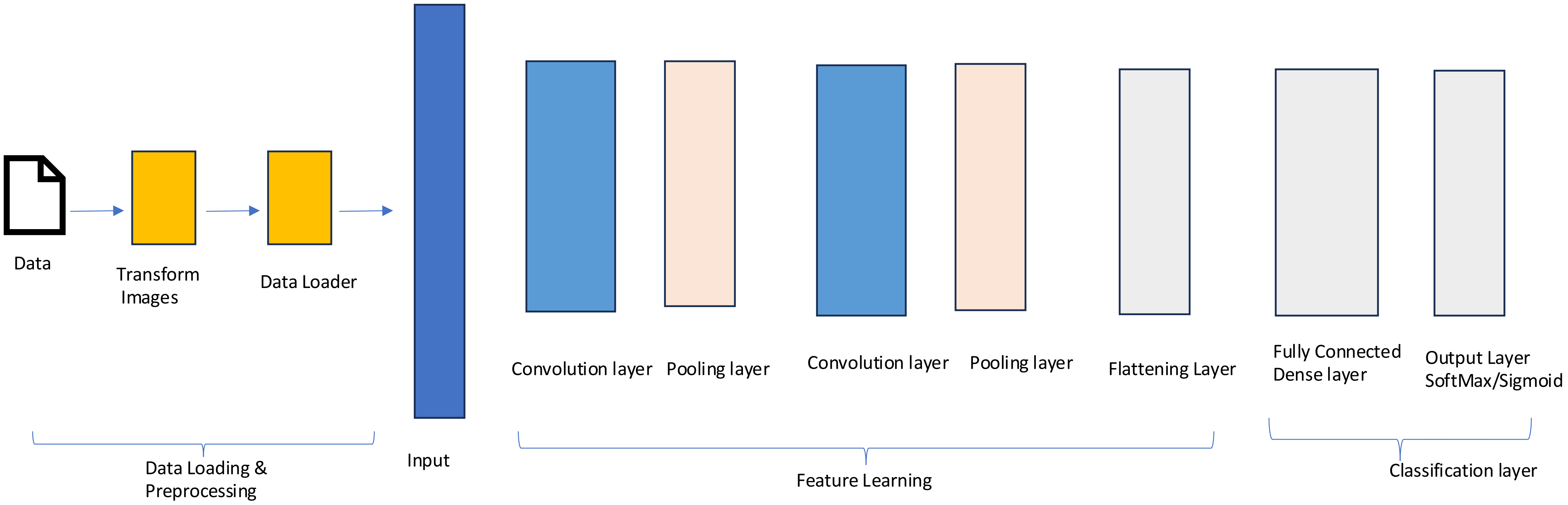
Sample No.	Actual Value	Predicted Value	Error
1	100	95	5
2	200	190	10
3	300	295	5
4	400	390	10
5	500	495	5

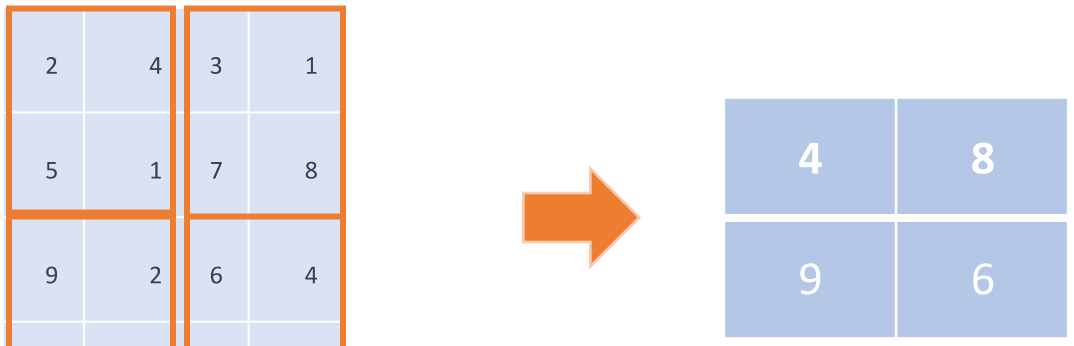

Input layer
Hidden layer
output Layer


Initialize Random Weights
Prediction on train data with random weights
Error=Actual Value-Predicted Value
Update Weights with respect to error

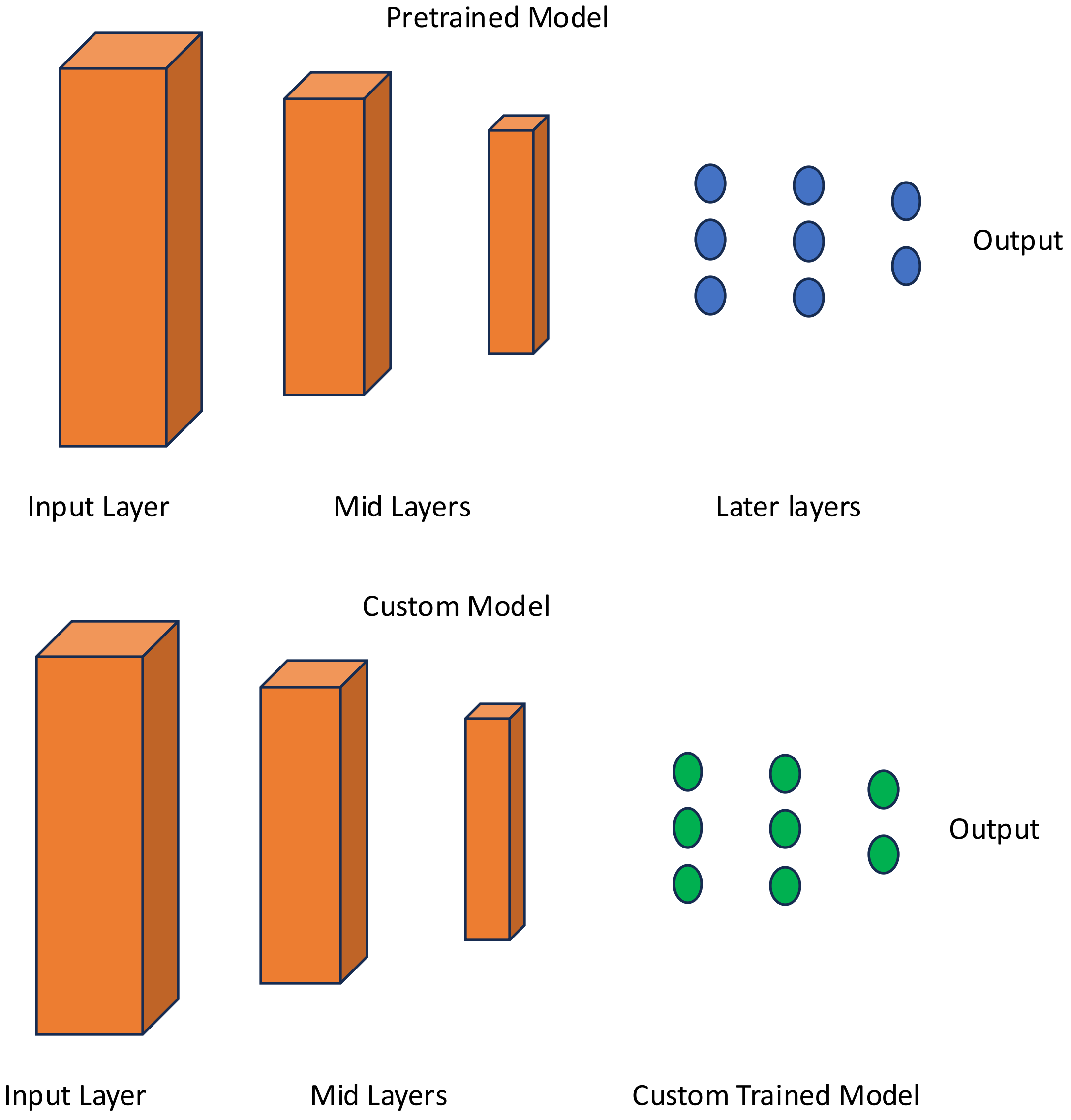

Convolution Filters
Feature Maps


0 1 0 1 0
1 1 0 1 1
0 1 0 0 0
0 1 0 0 1
1 0 1 1 0
Image
*
Multiple Convolution Filters
=
Feature Map

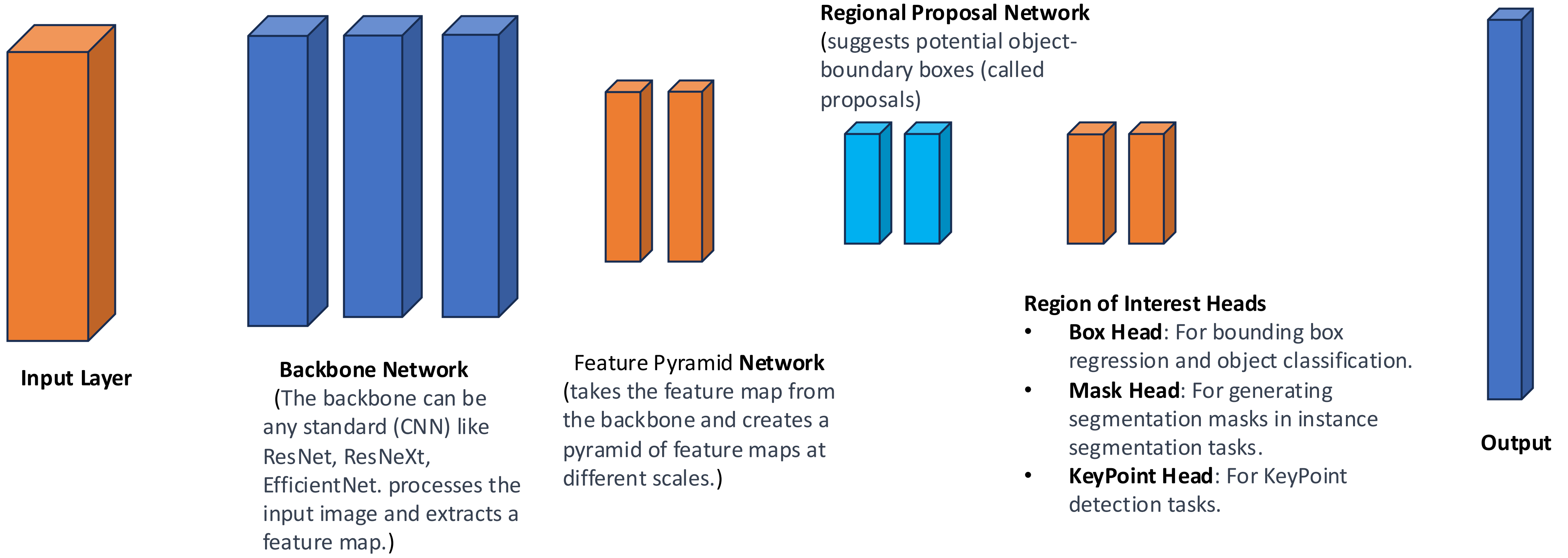

Data
Transform Images
Data Loader
Input
Convolution layer
Pooling layer
Convolution layer
Pooling layer
Flattening Layer
Fully Connected Dense layer
Output Layer SoftMax/Sigmoid
Data Loading & Preprocessing
Feature Learning
Classification layer



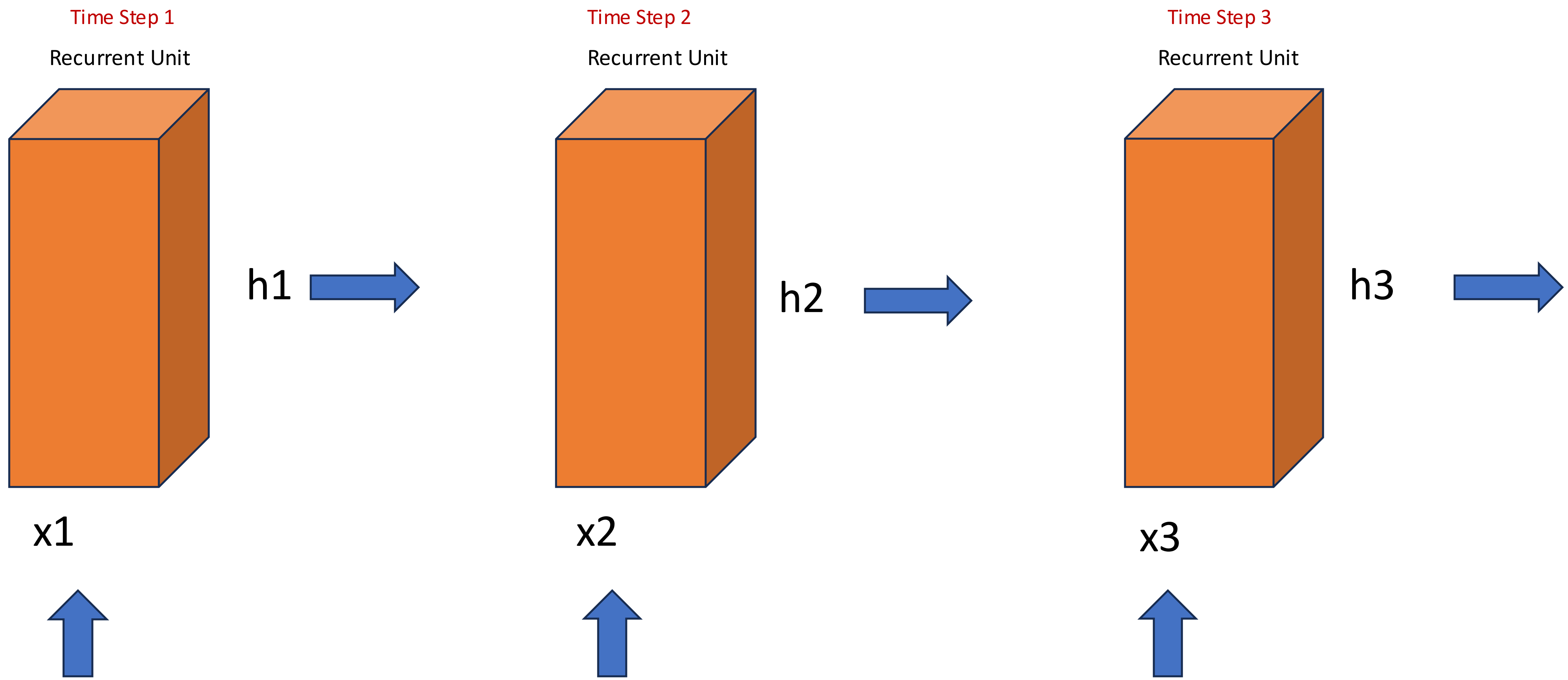
Max Pooling



Pretrained Model
Output
Input Layer
Mid Layers
Later layers
Custom Model
Output
Input Layer
Mid Layers
Custom Trained Model



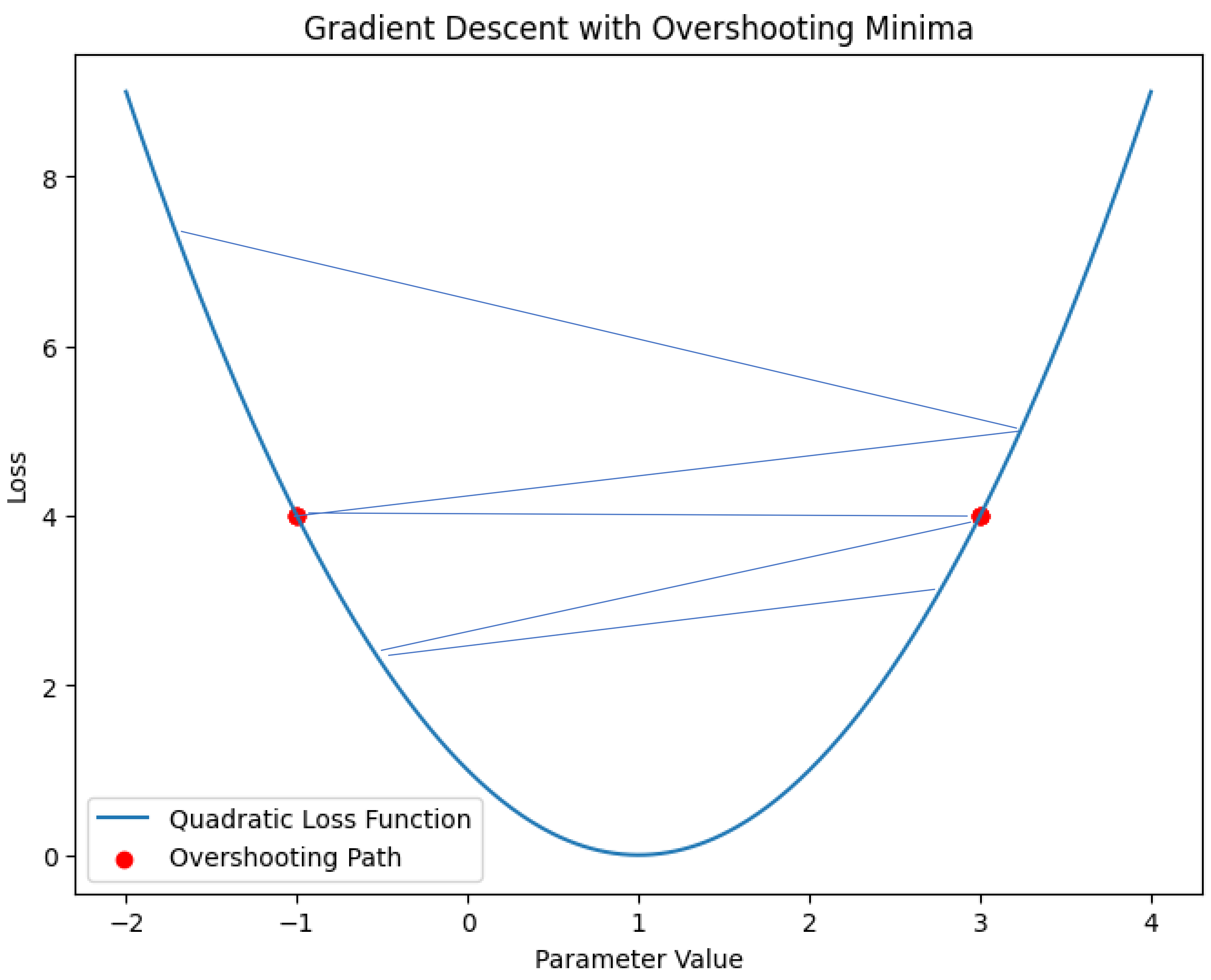
Input Layer
Backbone Network
(The backbone can be any standard (CNN) like ResNet, ResNeXt, EfficientNet. processes the input image and extracts a feature map.)
Feature Pyramid Network
(takes the feature map from the backbone and creates a pyramid of feature maps at different scales.)
Regional Proposal Network
(suggests potential object-boundary boxes (called proposals)
Region of Interest Heads
• Box Head: For bounding box regression and object classification.
• Mask Head: For generating segmentation masks in instance segmentation tasks.
• KeyPoint Head: For KeyPoint detection tasks.
Output



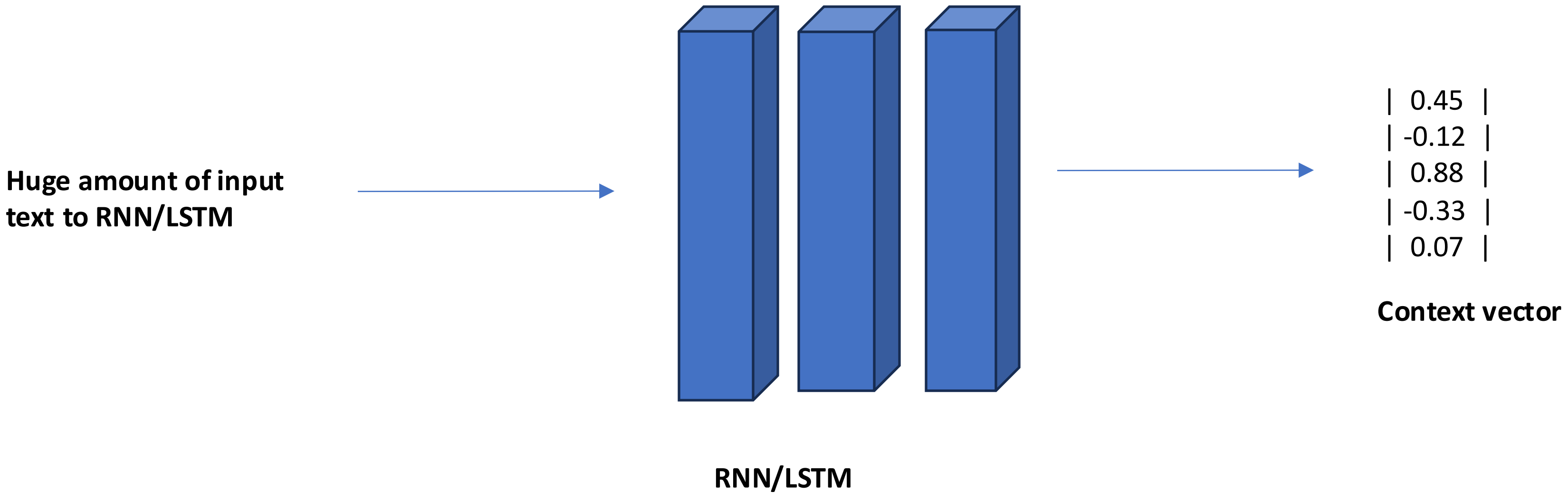
$$h1 = \tanh(Wh \cdot h0 + Wx \cdot x1)$$

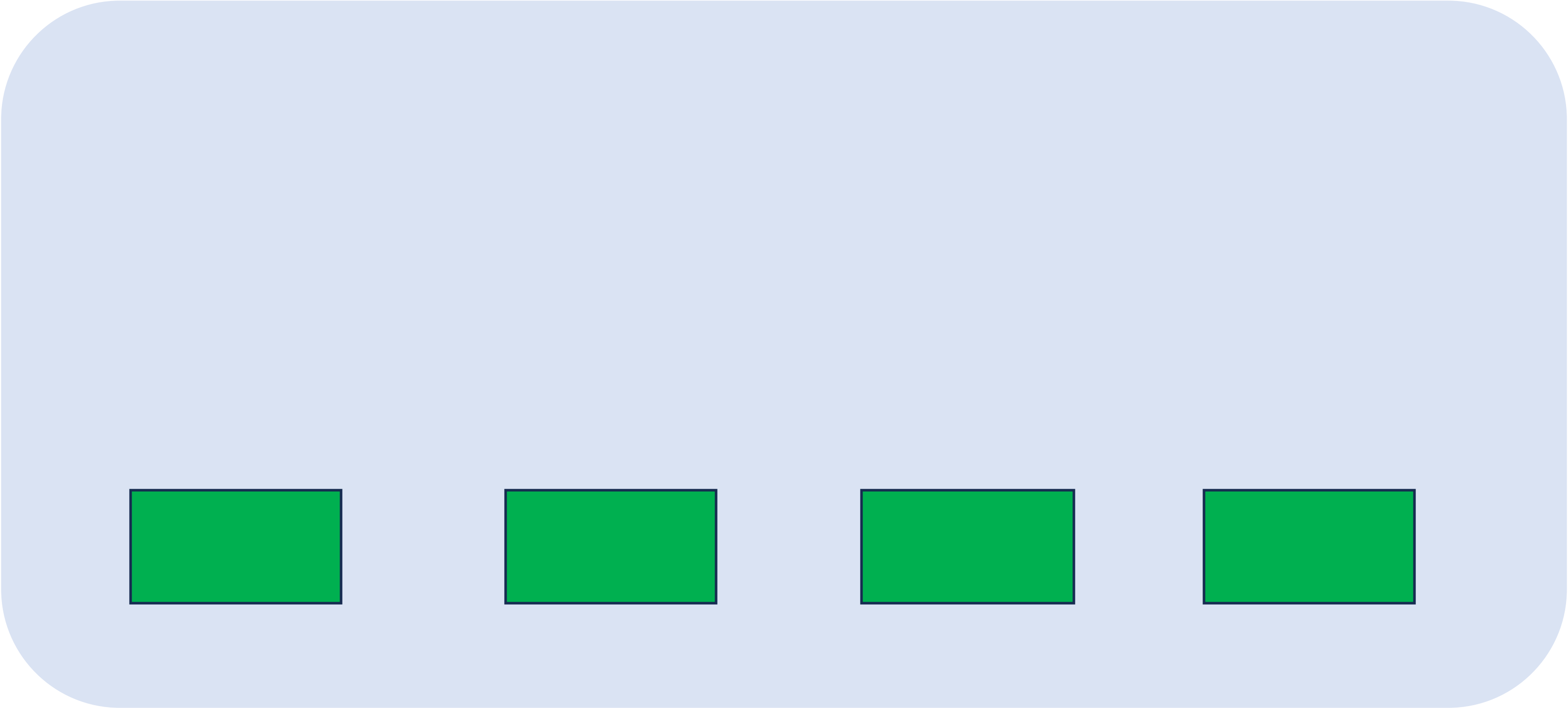
$$h2 = \tanh(Wh \cdot h1 + Wx \cdot x2)$$

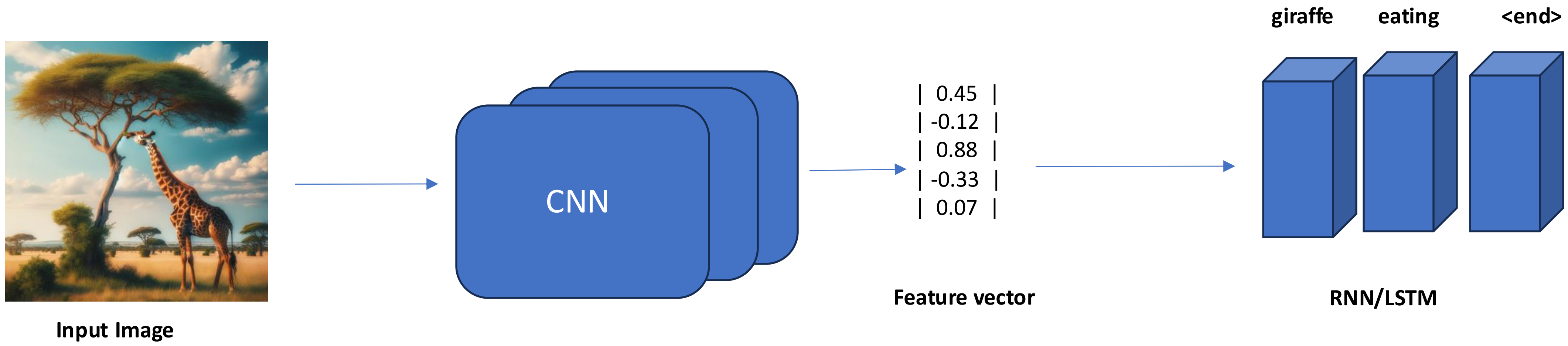
$$h3 = \tanh(Wh \cdot h2 + Wx \cdot x3)$$

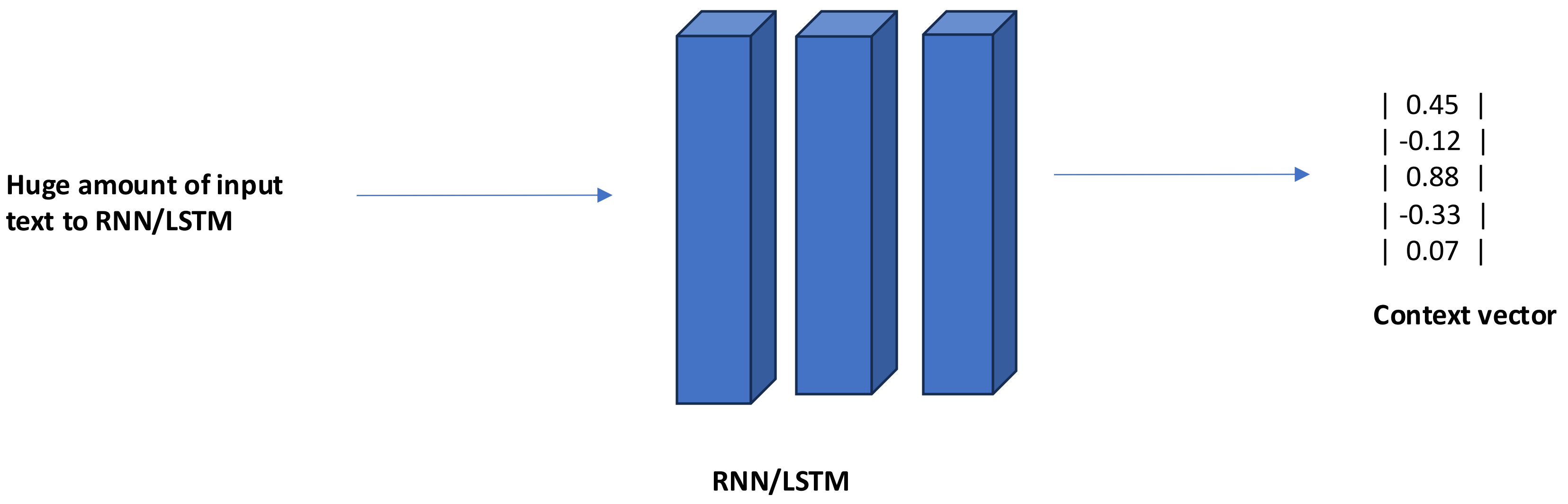


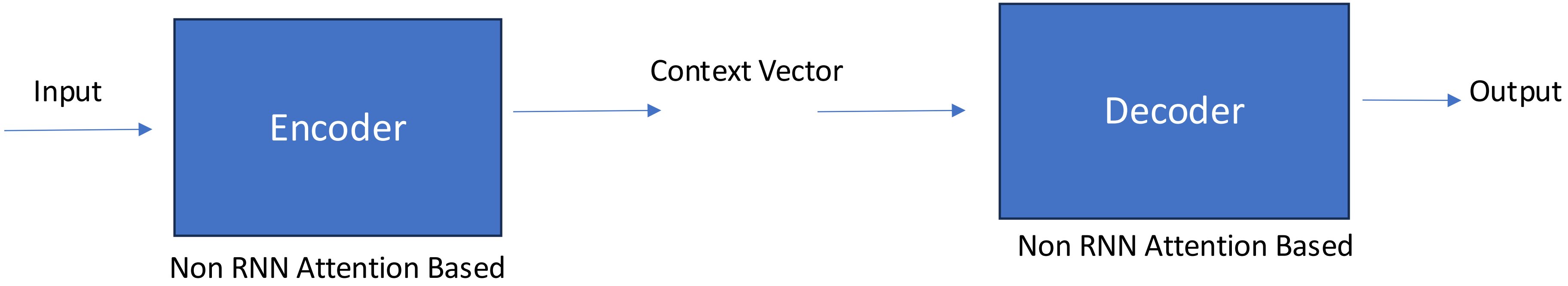
Gradient Descent with Overshooting Minima
Loss
Parameter Value
8
6
4
2
0
−2
−1
0
1
2
3
4
Quadratic Loss Function
Overshooting Path


Huge amount of input text to RNN/LSTM
RNN/LSTM
| 0.45 |
| -0.12 |
| 0.88 |
| -0.33 |
| 0.07 |
Context vector




giraffe
eating
<end>
CNN
| 0.45 |
| -0.12 |
| 0.88 |
| -0.33 |
| 0.07 |
Input Image
Feature vector
RNN/LSTM


Huge amount of input text to RNN/LSTM
RNN/LSTM
| 0.45 |
| -0.12 |
| 0.88 |
| -0.33 |
| 0.07 |
Context vector


Input
Encoder
Non RNN Attention Based
Context Vector
Decoder
Non RNN Attention Based
Output

Convolution Gan

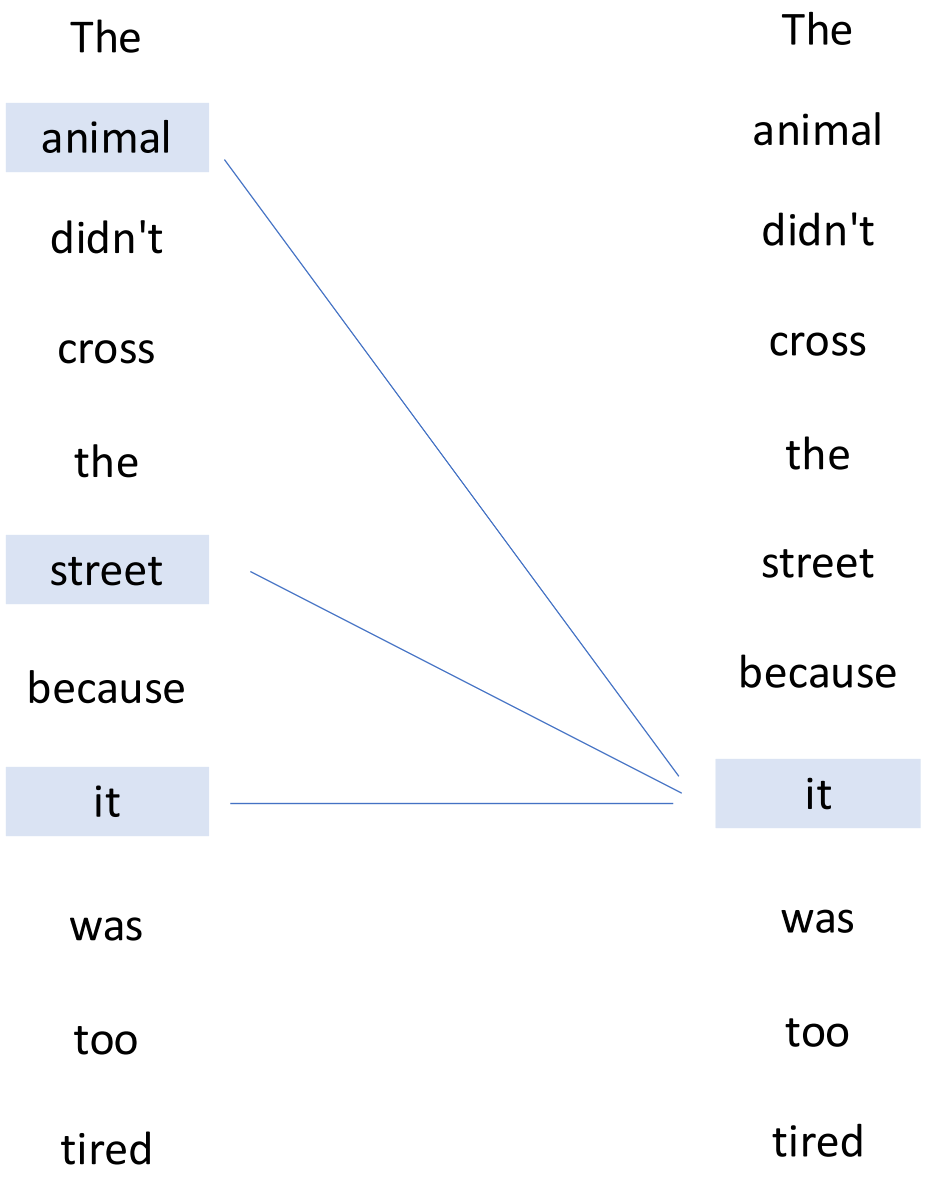
$$\begin{matrix} 1&1&1&0&0\\ 1&1&1&0&0\\ 1&1&1&0&0\\ 0&0&0&0&0\\ 0&0&0&0&0 \end{matrix} \quad X \quad \begin{matrix} 1&0&1\\ 0&1&0\\ 1&0&1 \end{matrix} \quad = \quad \begin{matrix} 4&3&4\\ 3&4&3\\ 4&3&4 \end{matrix}$$

Upscaling Matrix

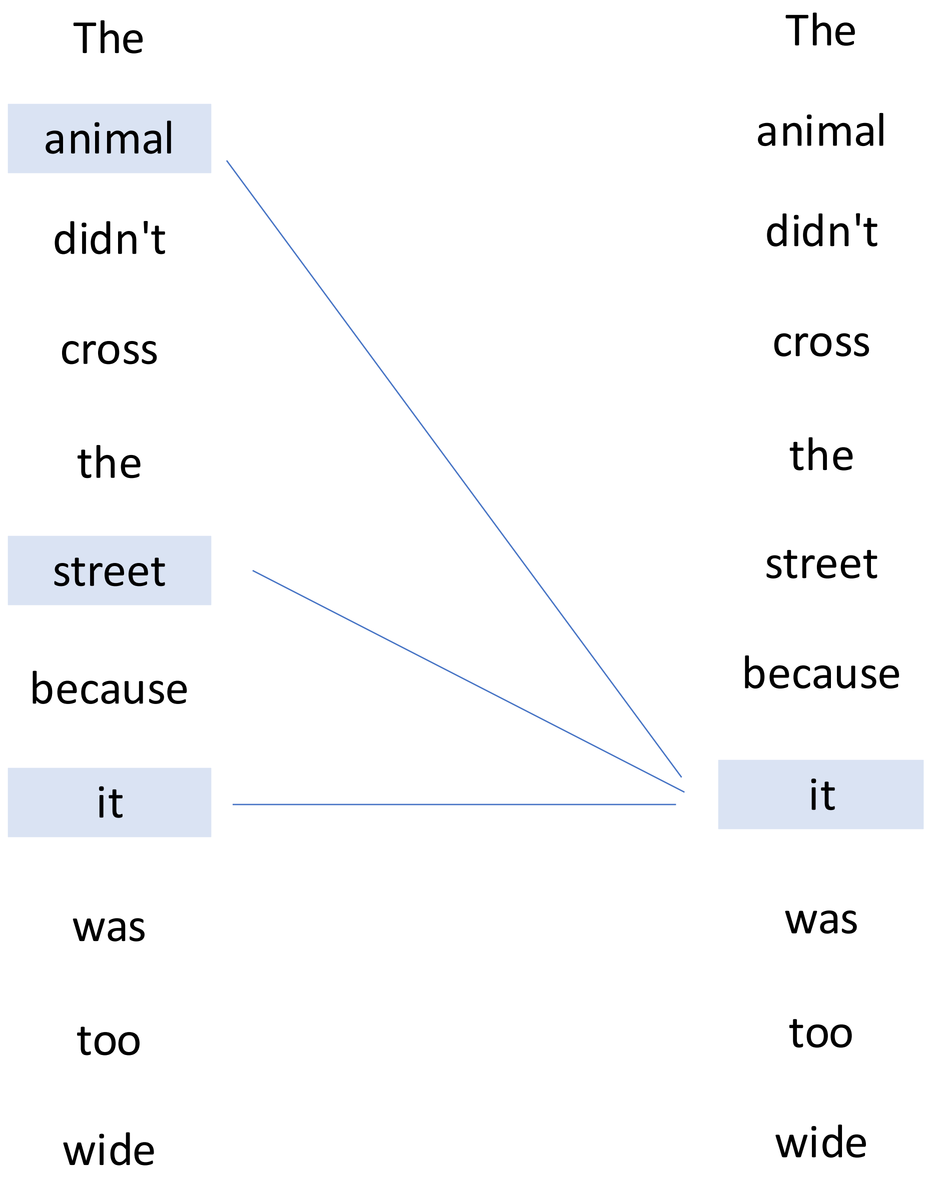
$$\begin{matrix} 1&2\\ 3&4 \end{matrix} \quad X \quad \begin{matrix} 1&0&1\\ 0&1&0\\ 1&0&1 \end{matrix} \quad = \quad \begin{matrix} 1&0&2&0&2\\ 0&1&0&2&0\\ 3&0&4&0&4\\ 0&3&0&4&0\\ 3&0&4&0&4 \end{matrix}$$

Downscaling Matrix

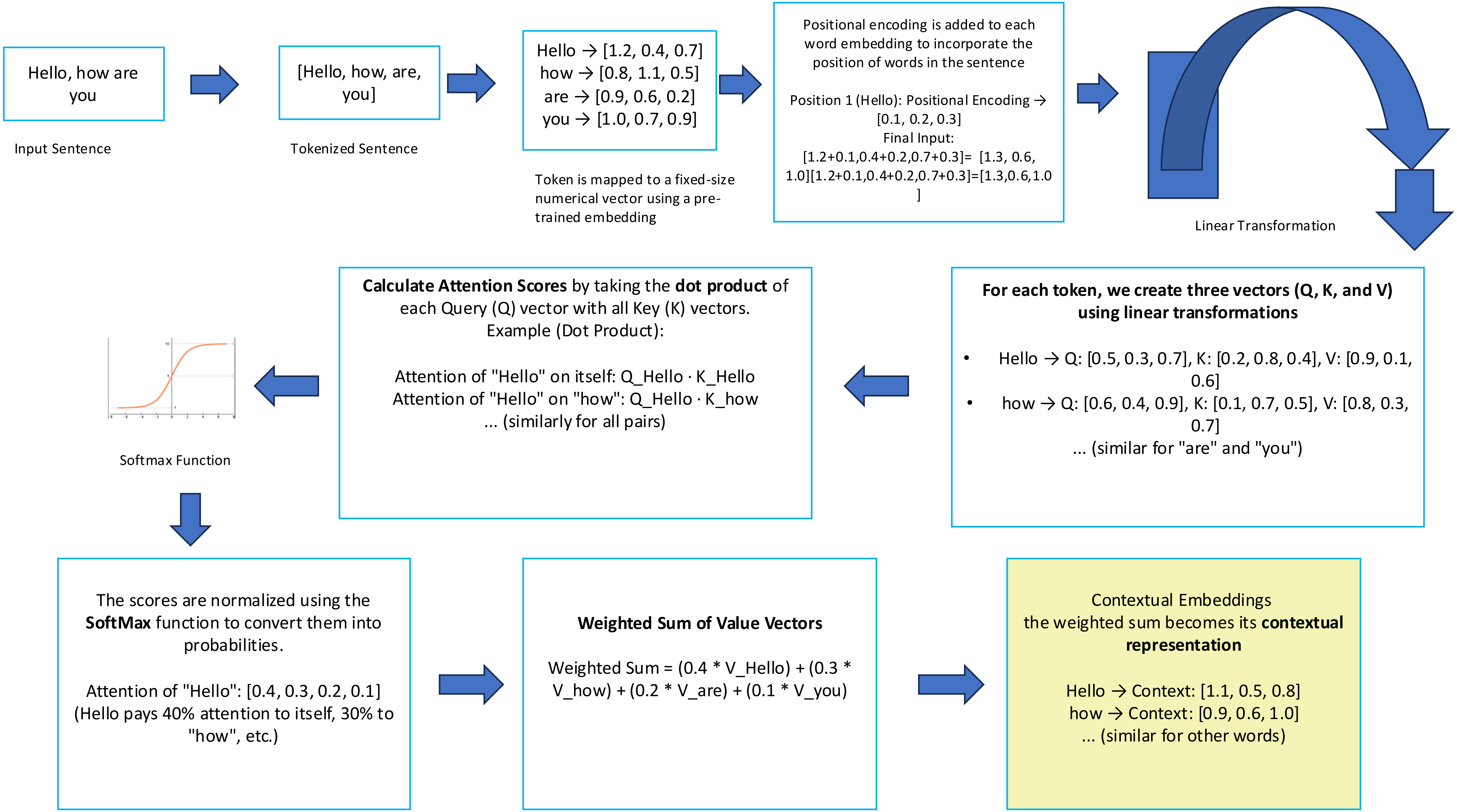
$$\begin{matrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{matrix} \quad X \quad \begin{matrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{matrix} \quad = \quad \begin{matrix} 4 & 4 \\ 4 & 4 \end{matrix}$$

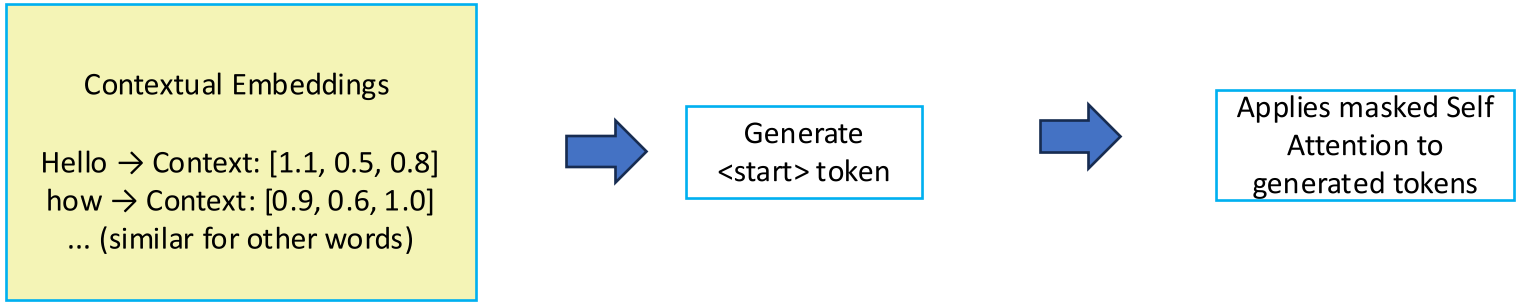


It is in reference to the animal.



It is in reference to the street.


Hello, how are you
Input Sentence
[Hello, how, are, you]
Tokenized Sentence
Hello → [1.2, 0.4, 0.7]
how → [0.8, 1.1, 0.5]
are → [0.9, 0.6, 0.2]
you → [1.0, 0.7, 0.9]
Token is mapped to a fixed-size numerical vector using a pre-trained embedding
Positional encoding is added to each word embedding to incorporate the position of words in the sentence
Position 1 (Hello): Positional Encoding → [0.1, 0.2, 0.3]
Final Input:
[1.2+0.1,0.4+0.2,0.7+0.3]= [1.3, 0.6, 1.0][1.2+0.1,0.4+0.2,0.7+0.3]=[1.3,0.6,1.0]
Linear Transformation
For each token, we create three vectors (Q, K, and V) using linear transformations
• Hello → Q: [0.5, 0.3, 0.7], K: [0.2, 0.8, 0.4], V: [0.9, 0.1, 0.6]
• how → Q: [0.6, 0.4, 0.9], K: [0.1, 0.7, 0.5], V: [0.8, 0.3, 0.7]
... (similar for "are" and "you")
Calculate Attention Scores by taking the dot product of each Query (Q) vector with all Key (K) vectors.
Example (Dot Product):
Attention of "Hello" on itself: Q_Hello · K_Hello
Attention of "Hello" on "how": Q_Hello · K_how
... (similarly for all pairs)
Softmax Function
The scores are normalized using the SoftMax function to convert them into probabilities.
Attention of "Hello": [0.4, 0.3, 0.2, 0.1]
(Hello pays 40% attention to itself, 30% to "how", etc.)
Weighted Sum of Value Vectors
Weighted Sum = (0.4 * V_Hello) + (0.3 * V_how) + (0.2 * V_are) + (0.1 * V_you)
Contextual Embeddings
the weighted sum becomes its contextual representation
Hello → Context: [1.1, 0.5, 0.8]
how → Context: [0.9, 0.6, 1.0]
... (similar for other words)


Contextual Embeddings
Hello → Context: [1.1, 0.5, 0.8]
how → Context: [0.9, 0.6, 1.0]
... (similar for other words)
Generate <start> token
Applies masked Self Attention to generated tokens