3011979 Intro to Deep Learning for Medical Imaging

L14: Recurrent neural network (and more encoder-decoder applications)

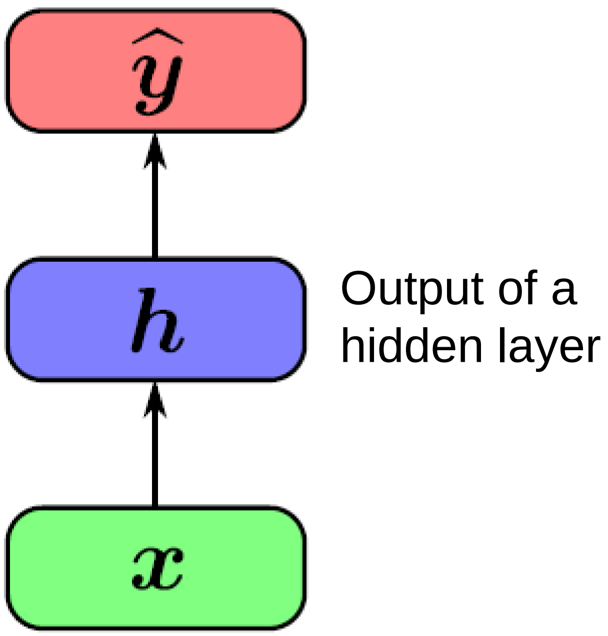
May 7th, 2021

Sira Sriswasdi, Ph.D.
Research Affairs, Faculty of Medicine
Chulalongkorn University

Recurrent neural network

Formulation of RNN

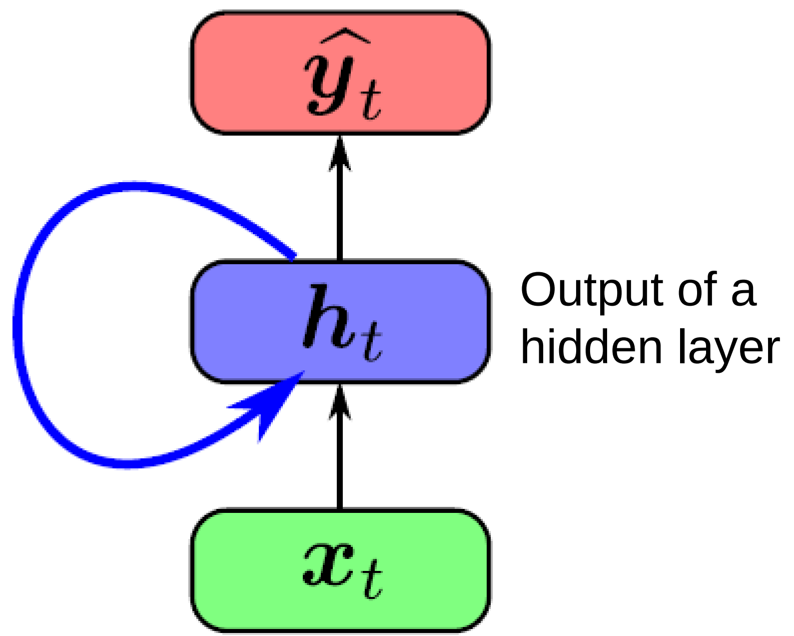
$$h = f(\boldsymbol{u} \cdot x + c)$$
$$\hat{y} = \boldsymbol{w} \cdot h + b$$



Fixed-length input

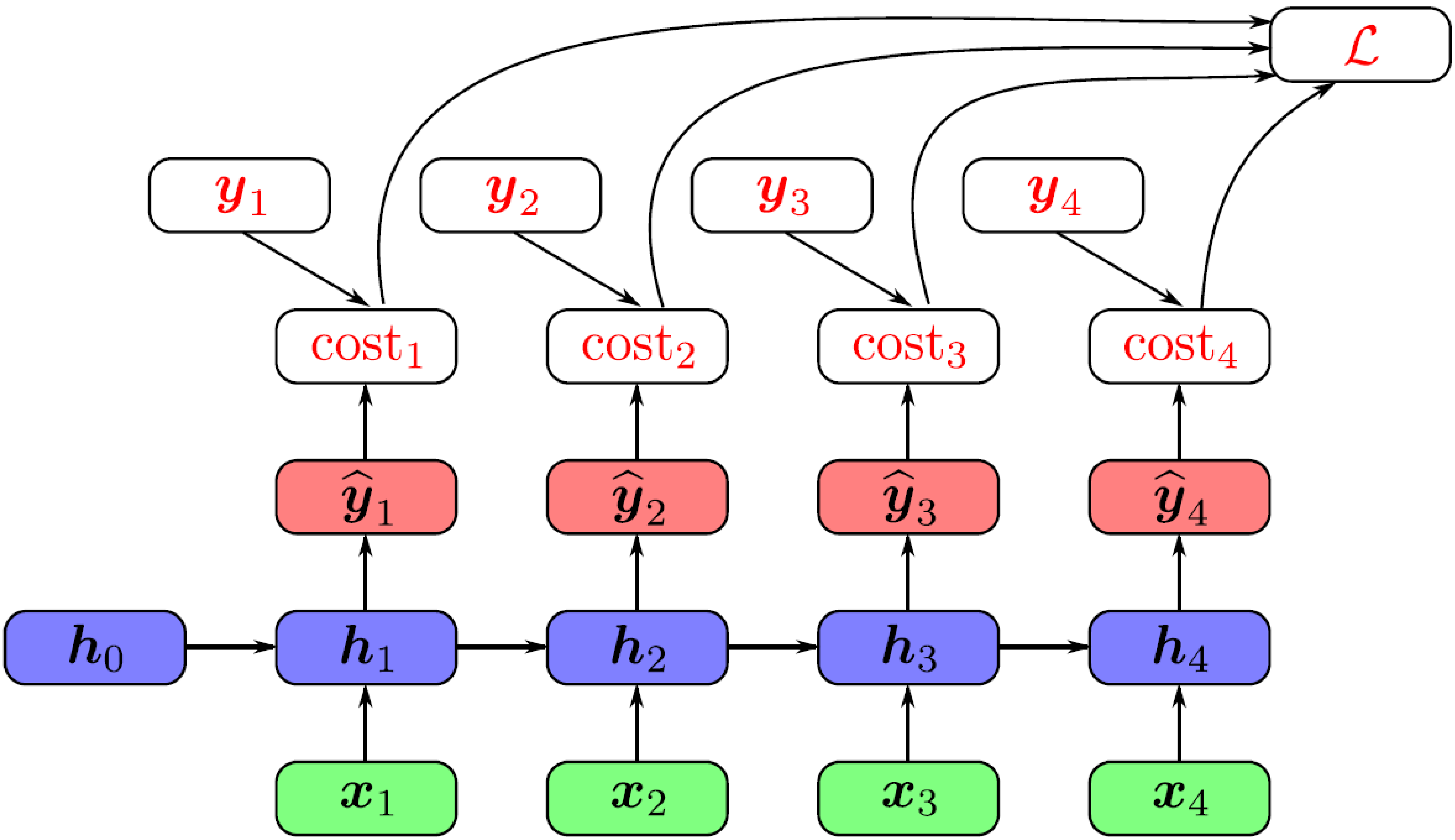
Shared weights!

$$h_1 = f(\boldsymbol{u} \cdot x_1 + \boldsymbol{v} \cdot h_0 + c)$$
$$h_2 = f(\boldsymbol{u} \cdot x_2 + \boldsymbol{v} \cdot h_1 + c)$$
$$\cdots$$
$$h_t = f(\boldsymbol{u} \cdot x_t + \boldsymbol{v} \cdot h_{t-1} + c)$$
$$\hat{y}_t = \boldsymbol{w} \cdot h_t + b$$



Variable-length input

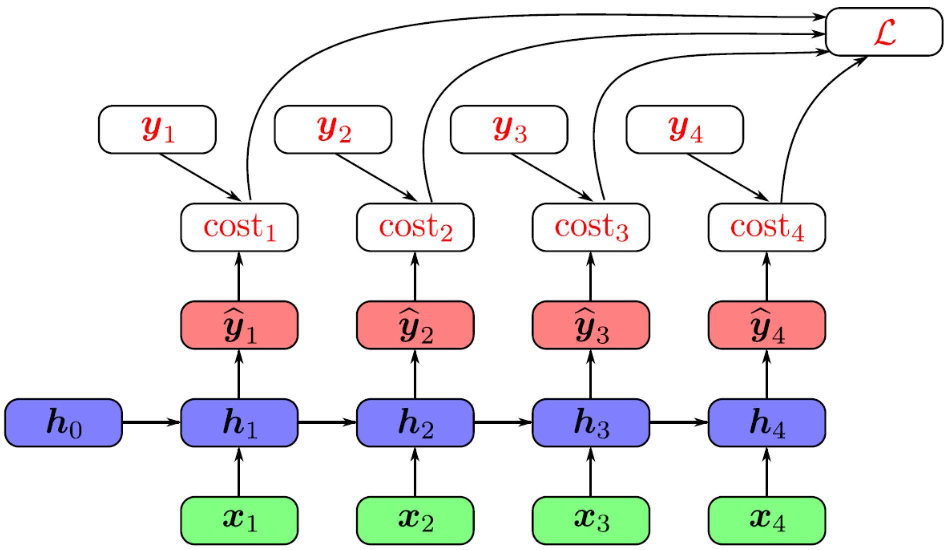
Gradient routes in RNN



- Loss function is evaluated at the end of the input sequence

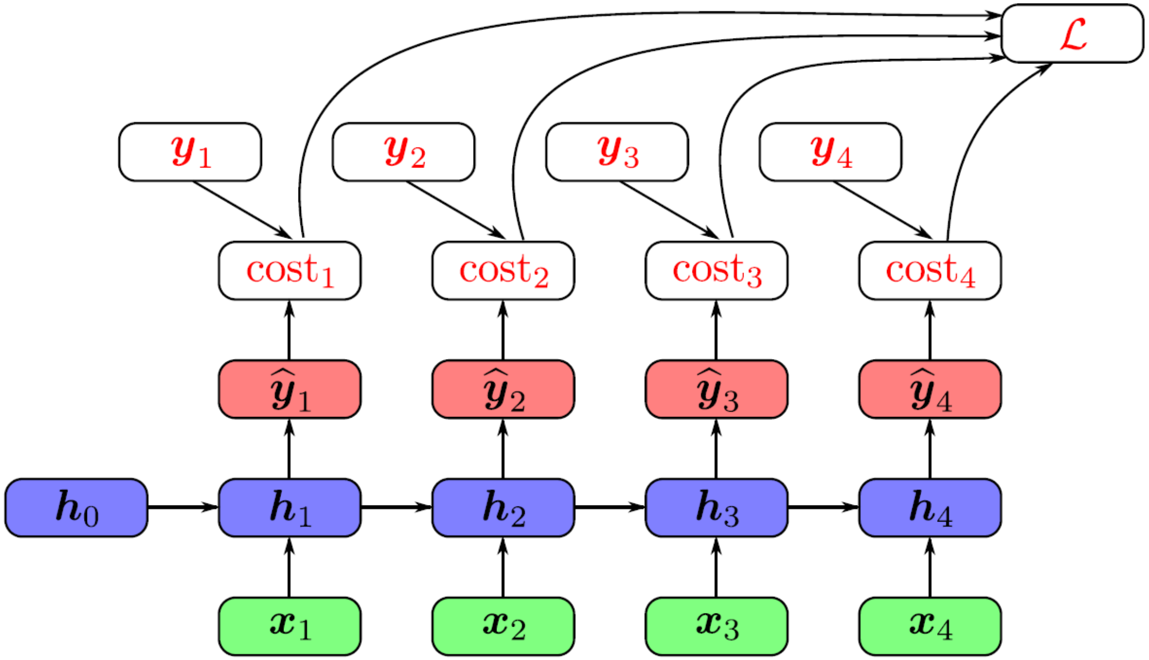
Backpropagation through time

$$h_1 = f(\boldsymbol{u} \cdot x_1 + \boldsymbol{v} \cdot h_0 + c)$$
$$h_2 = f(\boldsymbol{u} \cdot x_2 + \boldsymbol{v} \cdot h_1 + c)$$
$$\cdots$$
$$h_t = f(\boldsymbol{u} \cdot x_t + \boldsymbol{v} \cdot h_{t-1} + c)$$
$$\hat{y}_t = \boldsymbol{w} \cdot h_t + b$$



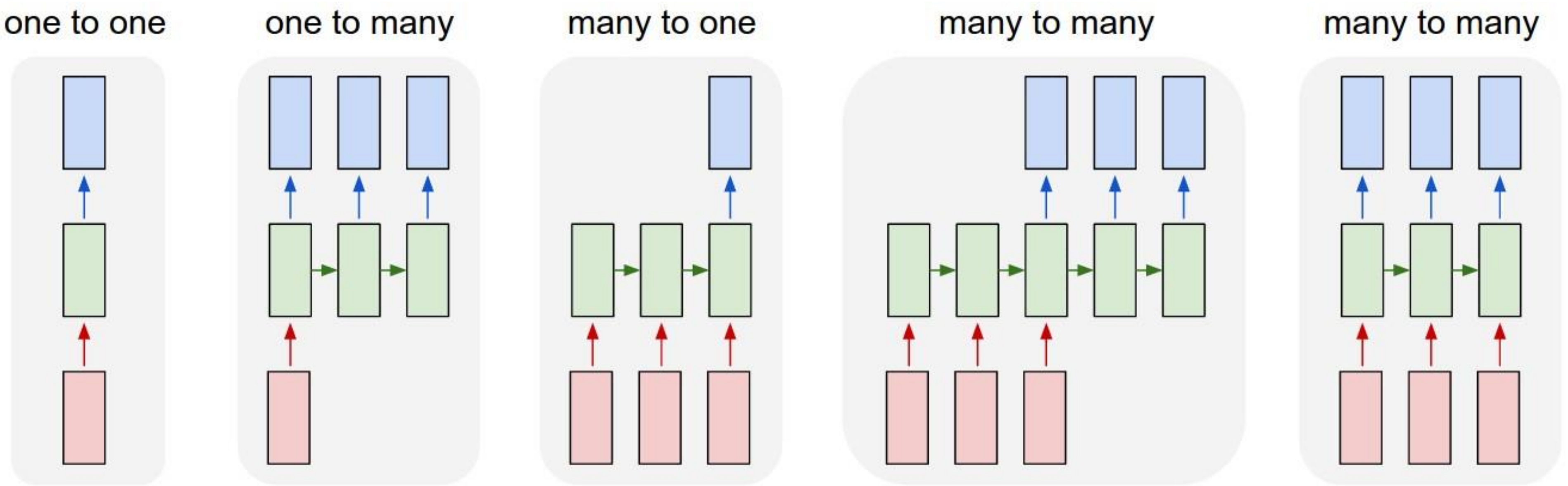
- Because weights $\boldsymbol{u}$ and $\boldsymbol{v}$ are shared, we backpropagate gradients through all h_i
  - $\frac{\delta L}{\delta u} = \frac{\delta L}{\delta h_1}\frac{\delta h_1}{\delta u} + \frac{\delta L}{\delta h_2}\frac{\delta h_2}{\delta u} + \frac{\delta L}{\delta h_3}\frac{\delta h_3}{\delta u} + \frac{\delta L}{\delta h_4}\frac{\delta h_4}{\delta u}$
  - $\frac{\delta L}{\delta v} = \frac{\delta L}{\delta h_1}\frac{\delta h_1}{\delta v} + \frac{\delta L}{\delta h_2}\frac{\delta h_2}{\delta v} + \frac{\delta L}{\delta h_3}\frac{\delta h_3}{\delta v} + \frac{\delta L}{\delta h_4}\frac{\delta h_4}{\delta v}$

Conditional probability view of RNN



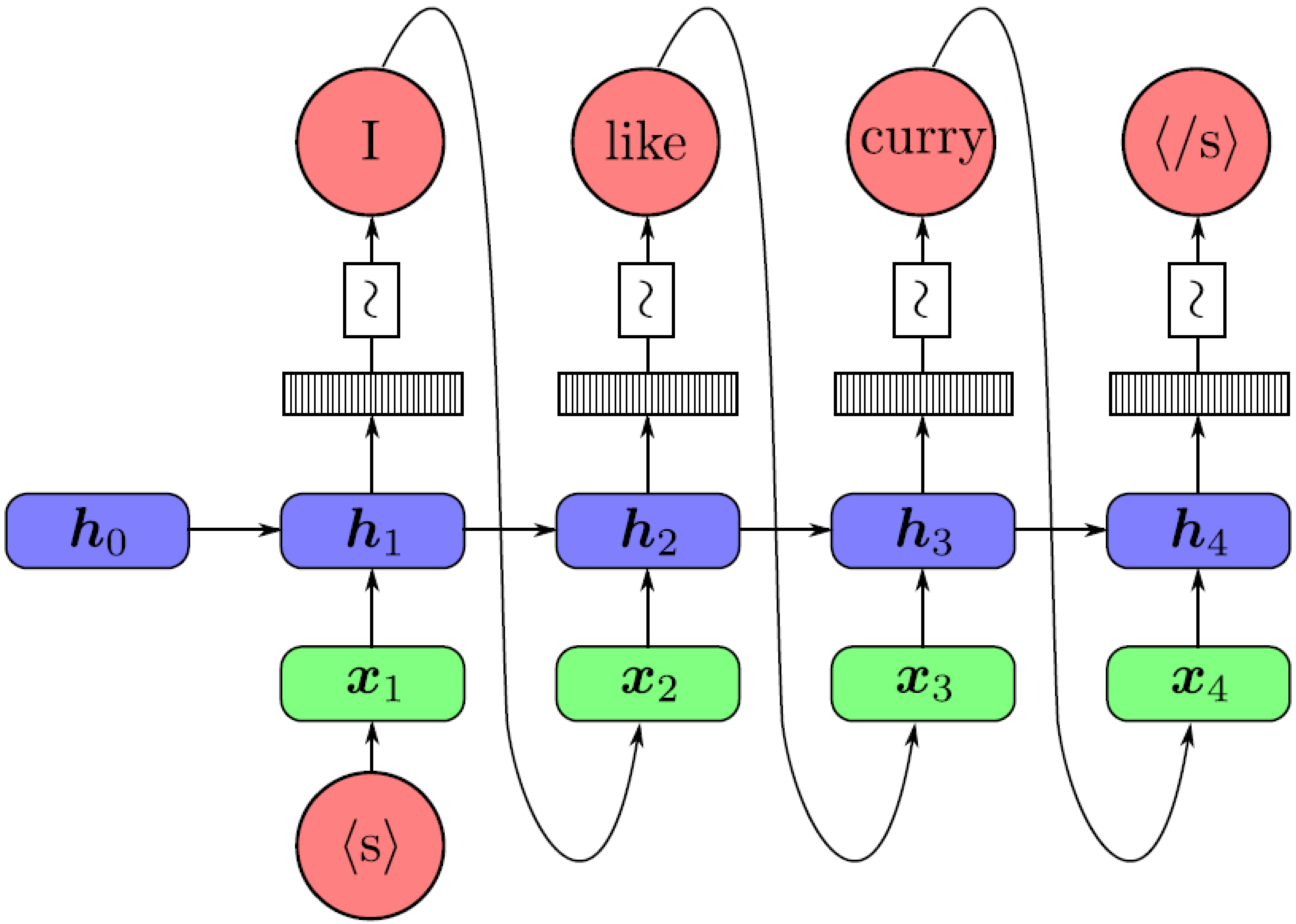
- $\mathrm{P}(y_1, y_2, y_3, y_4) = \mathrm{P}(y_4|y_1, y_2, y_3)\mathrm{P}(y_3|y_1, y_2)\mathrm{P}(y_2|y_1)\ \mathrm{P}(y_1)$
- $y_t = \boldsymbol{w} \cdot h_t + b$
- $h_t = f(\boldsymbol{u} \cdot x_t + \boldsymbol{v} \cdot h_{t-1} + c)$
- h_t encodes the history of prior information
- $\boldsymbol{w}$ represents the conditional probability function

Applications of RNN

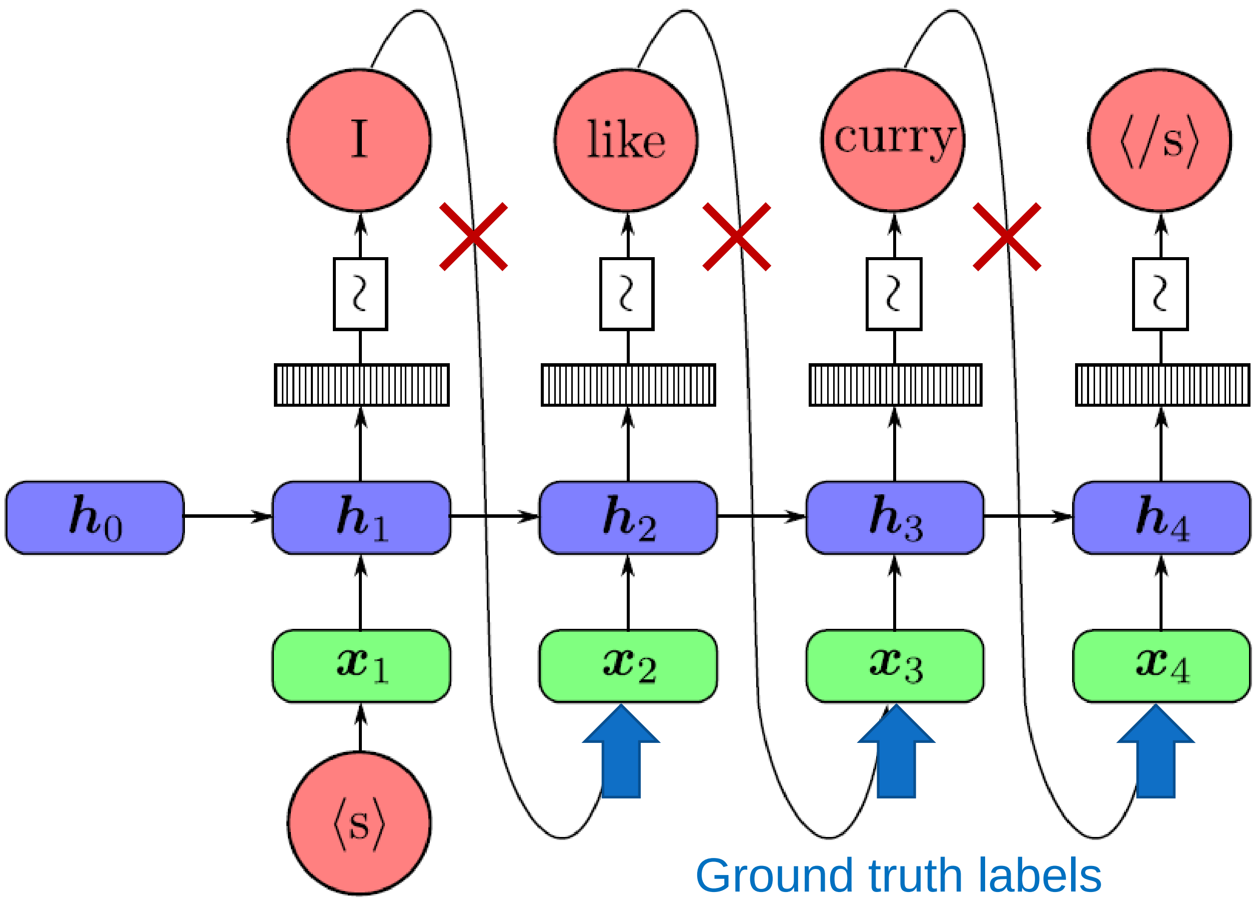


- Generate medical report = variable output length
- Extract keywords from written report = variable input length
- Translate sentence = variable input and output lengths

One-to-many: Auto-regressive approach

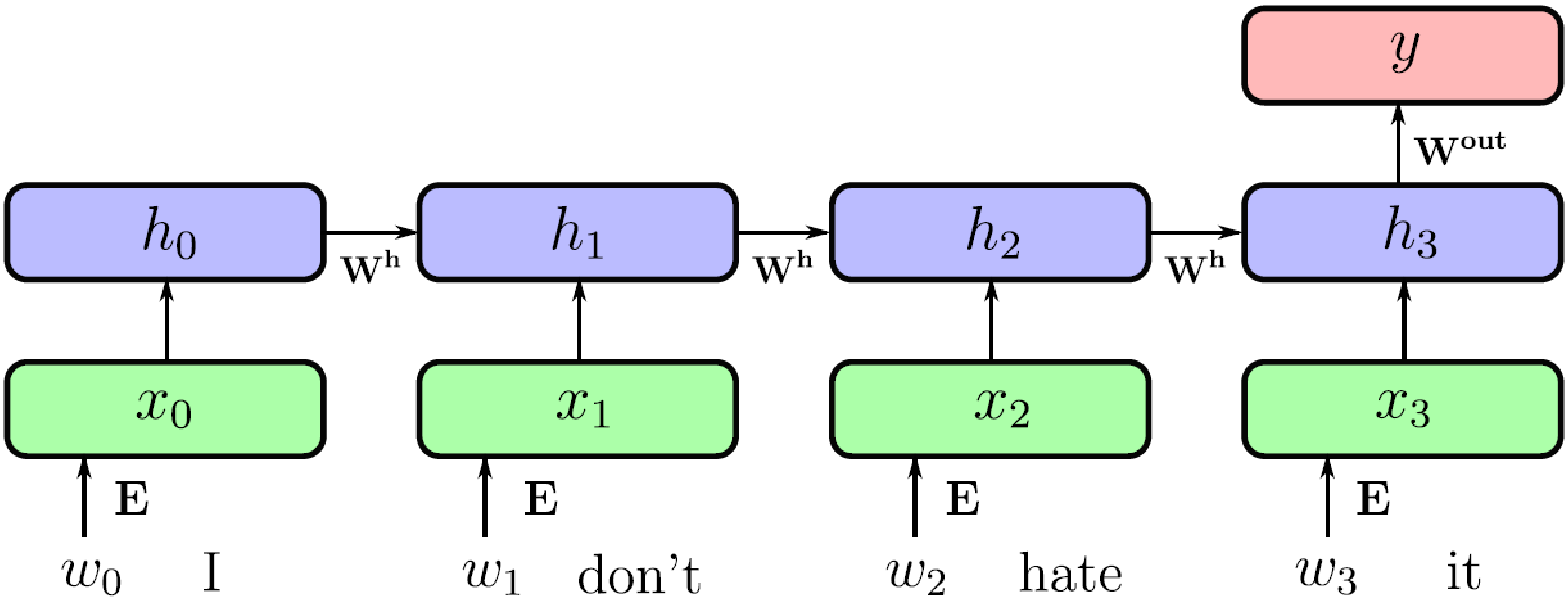


Teacher forcing technique



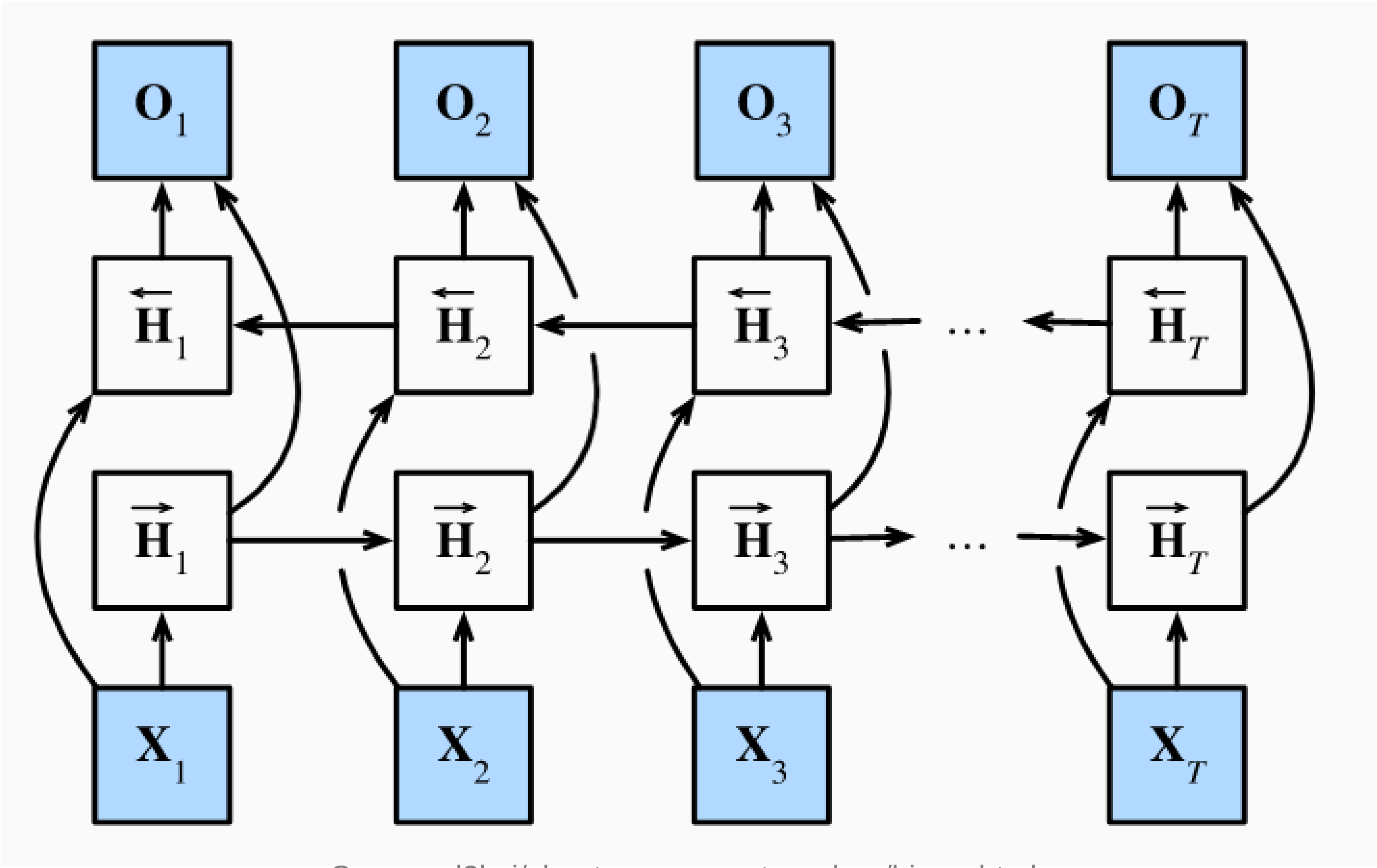
- During training of auto-regressive model, we feed the ground truth label instead of previous output
  - But during real usage, the previous output might be incorrect!

Many-to-one: Classification



- Simplest approach is to use the output of the last node
  - h_t encodes all prior information
  - But the impact from earlier input decrease over the steps
- Pool all the outputs
- Bidirectional model!

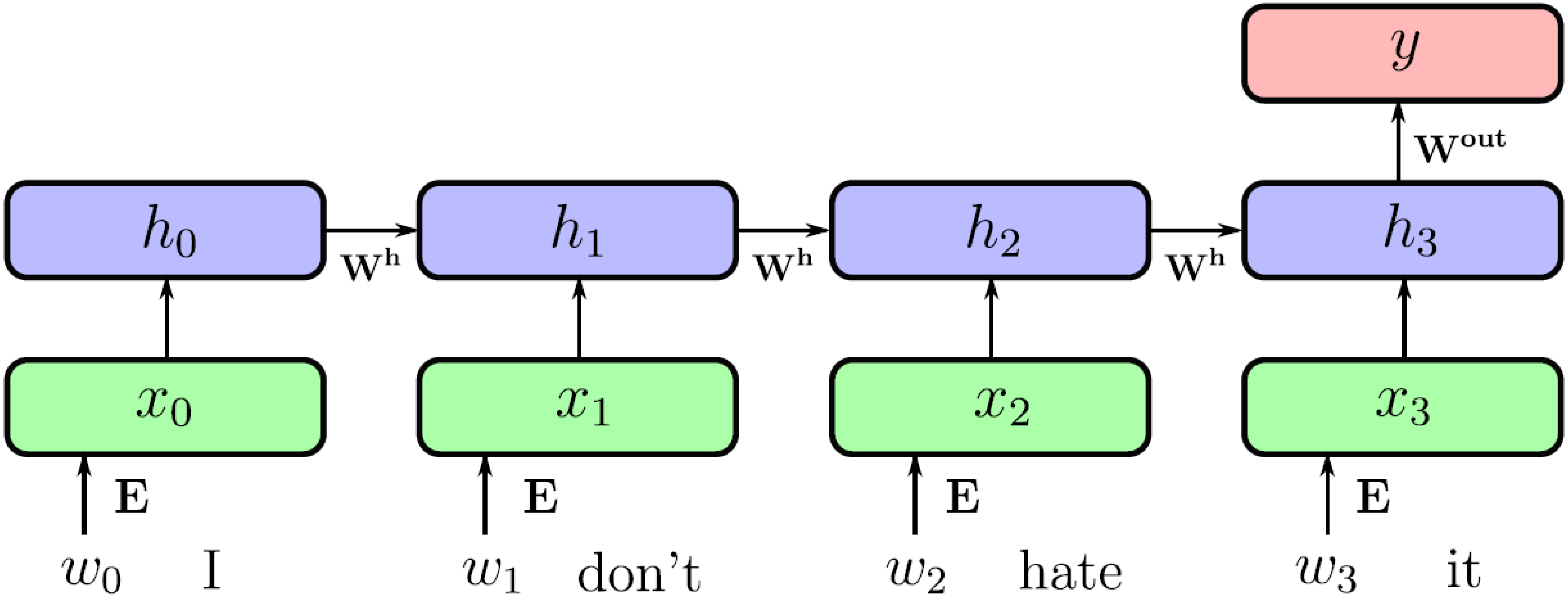
Bidirectional model



Source: d2l.ai/chapter_recurrent-modern/bi-rnn.html

- Just as RNN can learn h_t from h_{t-1} and x_t, it can also learn h_{t-1} from h_t and x_{t-1}
- Combine information from the forward and backward passes

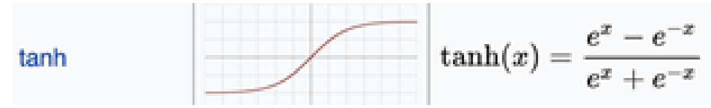
Vanishing gradient problem in RNN

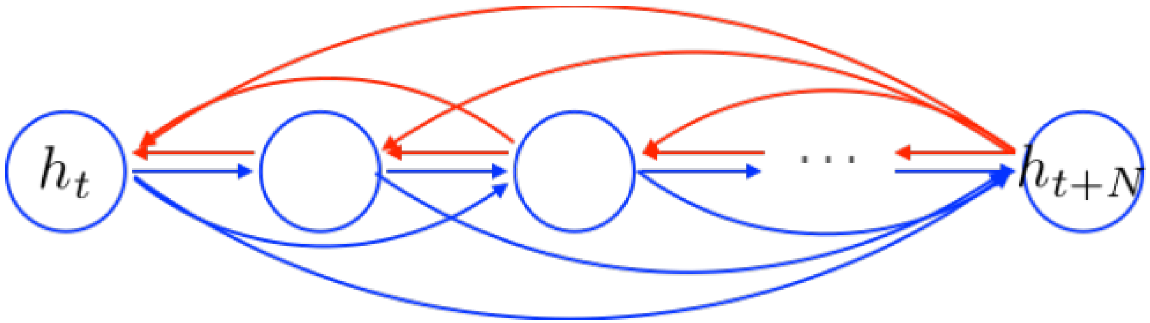


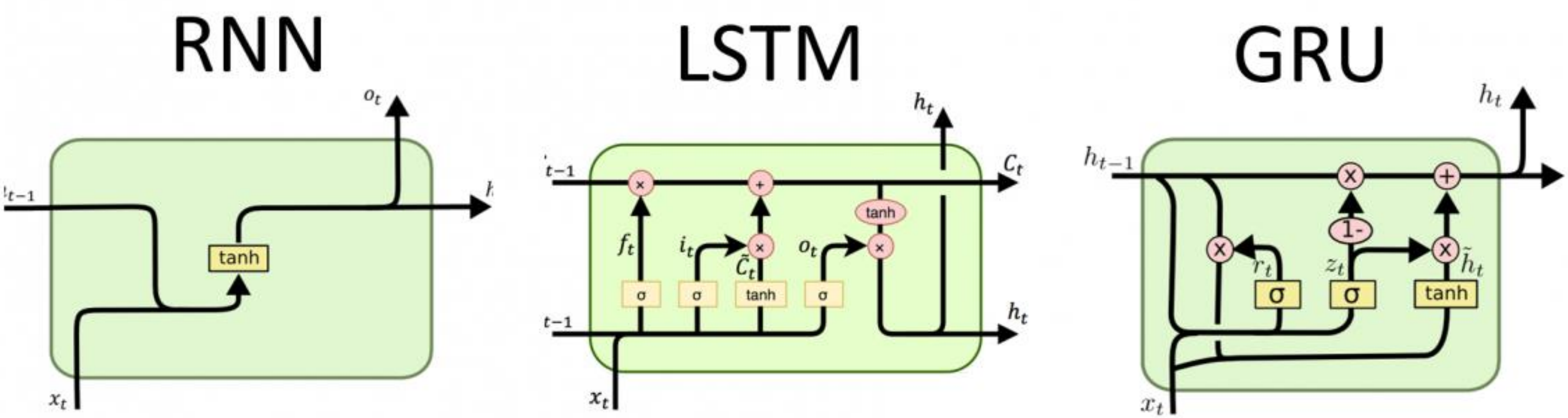
- Gradient is unraveled all the way to the first input
  - $\frac{\delta L}{\delta u} = \frac{\delta L}{\delta h_3}\frac{\delta h_3}{\delta h_2}\frac{\delta h_2}{\delta h_1}\frac{\delta h_1}{\delta h_0}\frac{\delta h_0}{\delta u}$
  - Number of terms = length of the input

RNN units

Adding "gates" to RNN



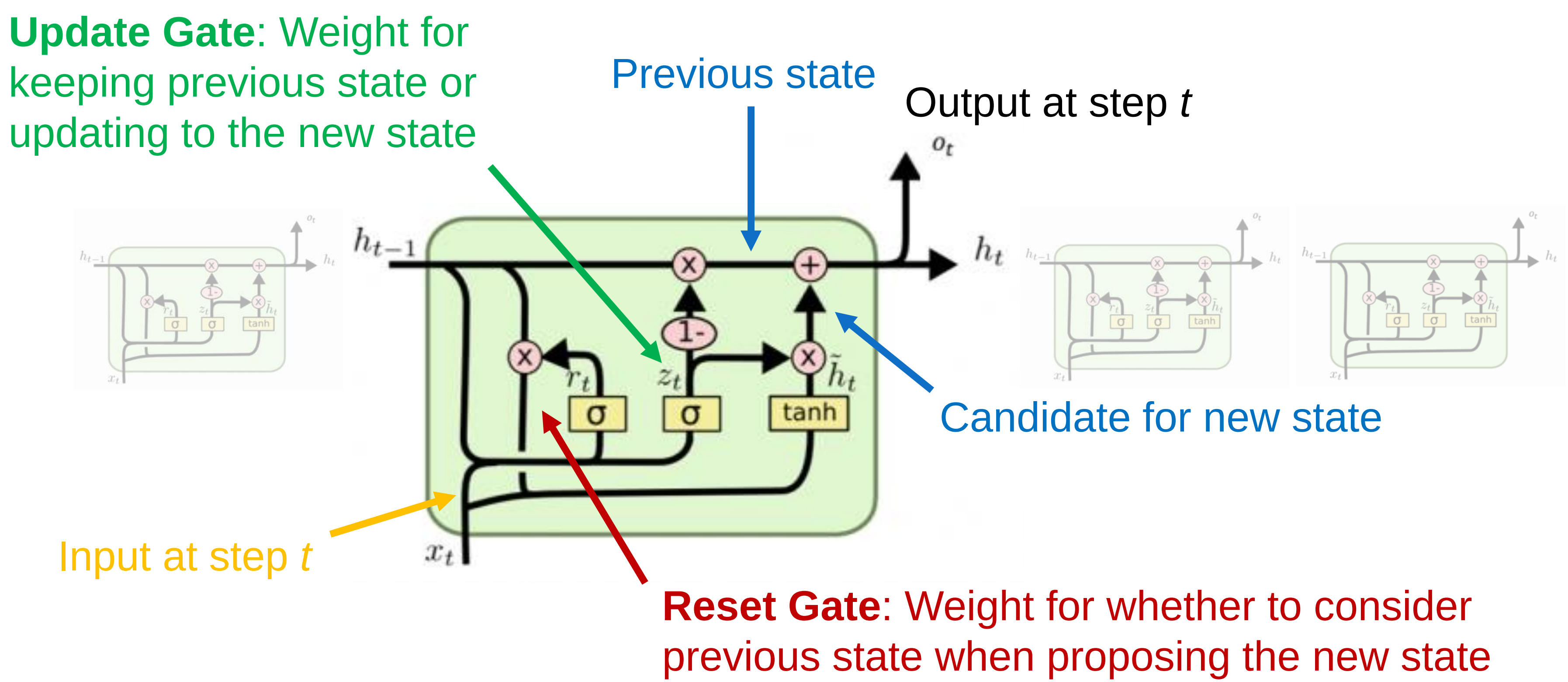




Source: www.linkedin.com/pulse/recurrent-neural-networks-rnn-gated-units-gru-long-short-robin-kalia

- Add a direct path from previous unit to the next
  - Allow the network to "forget" the past or "remember" it
  - h_t can be updated with x_t or stay the same as h_{t-1}

Gated Recurrent Unit (GRU)



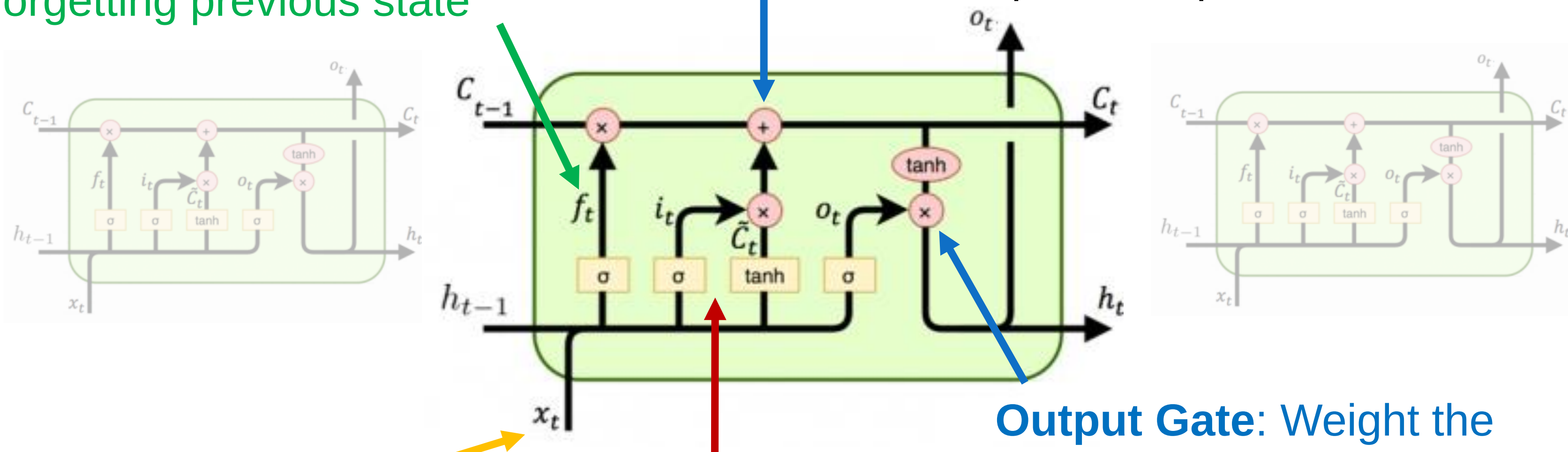
- New input, together with previous state, is used to determine whether to remember or forget the previous state
- Gradient can flow through the $h_{t-1} \rightarrow h_t$ shortcut

Long Short-Term Memory (LSTM)

Forget Gate: Weight for forgetting previous state

New "cell" state

Output at step *t*



Output Gate: Weight the contribution of current input on the current output

Input at step *t*

Input Gate: Propose new "cell" state

- Cell state C_t can carry memory over long term
- The contribution of current input on the output at step *t* can be assigned through the Output Gate

RNN in TensorFlow

1. `keras.layers.SimpleRNN`, a fully-connected RNN where the output from previous timestep is to be fed to next timestep.
2. `keras.layers.GRU`, first proposed in Cho et al., 2014.
3. `keras.layers.LSTM`, first proposed in Hochreiter & Schmidhuber, 1997.

```
model = keras.Sequential()
# Add an Embedding layer expecting input vocab of size 1000, and
# output embedding dimension of size 64.
model.add(layers.Embedding(input_dim=1000, output_dim=64))

# Add a LSTM layer with 128 internal units.
model.add(layers.LSTM(128))

# Add a Dense layer with 10 units.
model.add(layers.Dense(10))

model.summary()
```

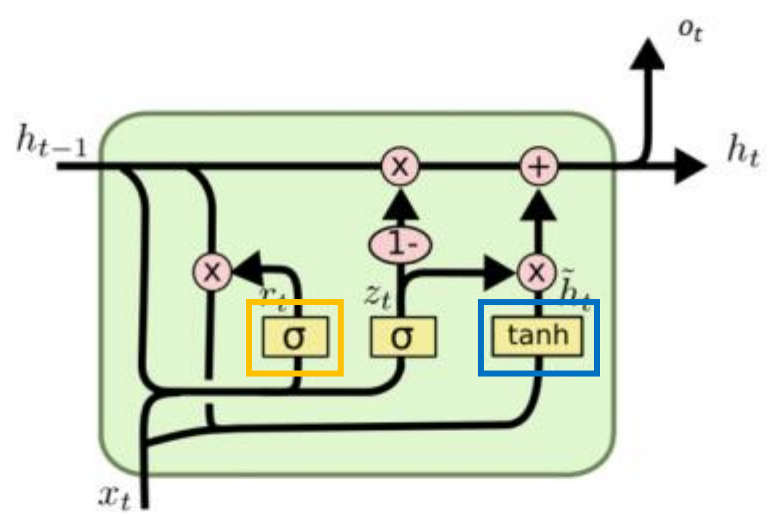
Input = (batch, time, data)

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 64)          64000
_________________________________________________________________
lstm (LSTM)                  (None, 128)               98816
_________________________________________________________________
dense (Dense)                (None, 10)                1290
=================================================================
Total params: 164,106
```

GRU in TensorFlow

```
tf.keras.layers.GRU(
    units, activation='tanh', recurrent_activation='sigmoid',
    use_bias=True, kernel_initializer='glorot_uniform',
    recurrent_initializer='orthogonal',
    bias_initializer='zeros', kernel_regularizer=None,
    recurrent_regularizer=None, bias_regularizer=None, activity_regularizer=None,
    kernel_constraint=None, recurrent_constraint=None, bias_constraint=None,
    dropout=0.0, recurrent_dropout=0.0, return_sequences=False, return_state=False,
    go_backwards=False, stateful=False, unroll=False, time_major=False,
    reset_after=True, **kwargs
)
```

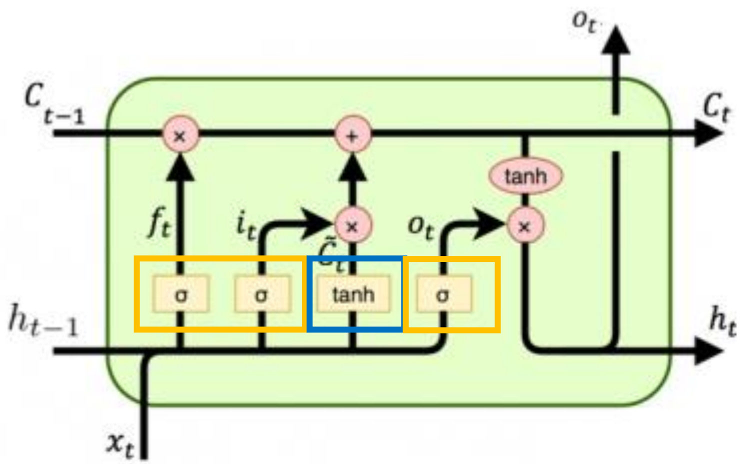
- Units = number of hidden neurons for the fully-connected layer that takes in h_{t-1} and x_t
- Set **return_sequence = True** if the model should output value at every step



LSTM in TensorFlow

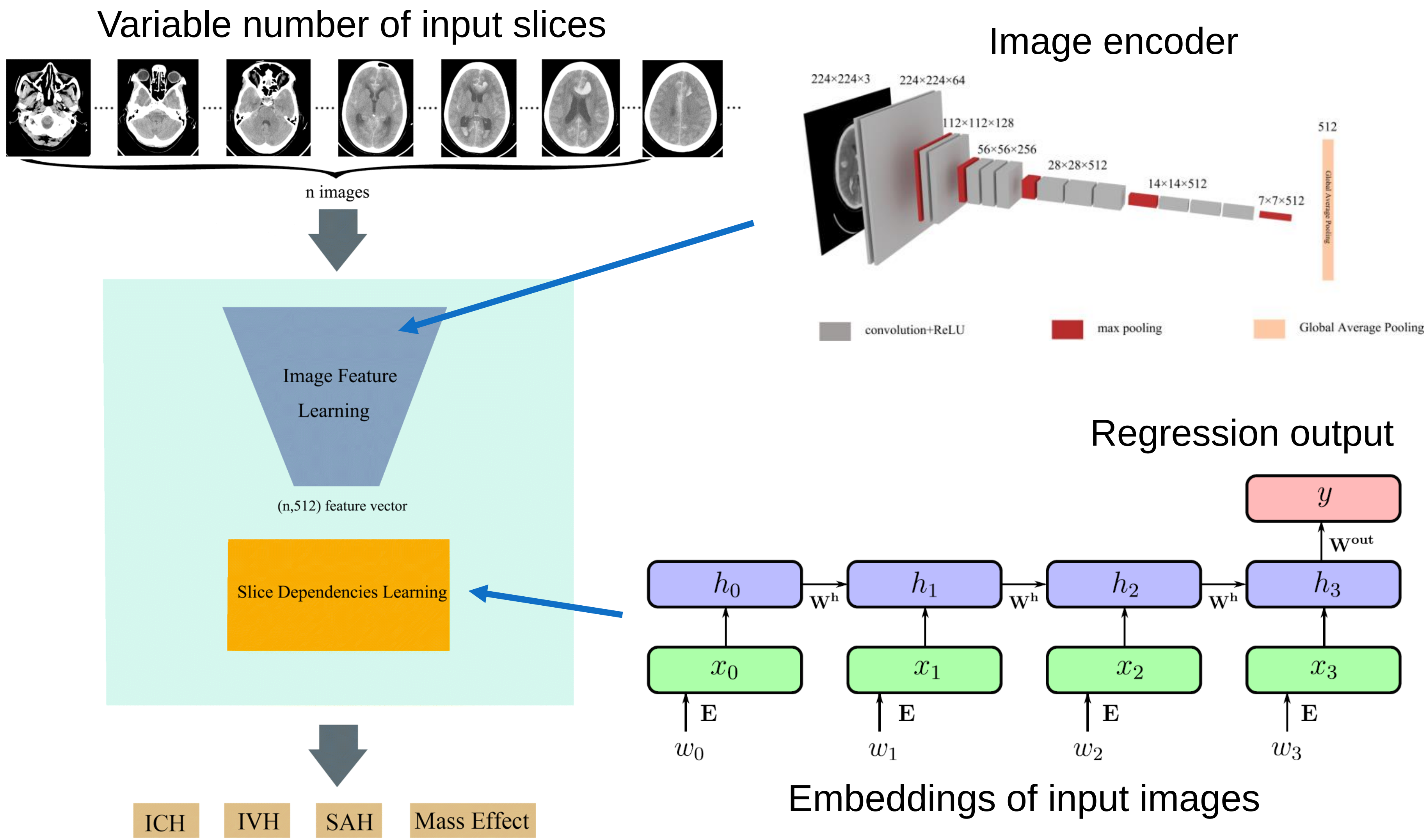
```
tf.keras.layers.LSTM(
    units, activation='tanh', recurrent_activation='sigmoid',
    use_bias=True, kernel_initializer='glorot_uniform',
    recurrent_initializer='orthogonal',
    bias_initializer='zeros', unit_forget_bias=True,
    kernel_regularizer=None, recurrent_regularizer=None, bias_regularizer=None,
    activity_regularizer=None, kernel_constraint=None, recurrent_constraint=None,
    bias_constraint=None, dropout=0.0, recurrent_dropout=0.0,
    return_sequences=False, return_state=False, go_backwards=False, stateful=False,
    time_major=False, unroll=False, **kwargs
)
```

- All gates share the same output layer and activation function (recurrent_activation)



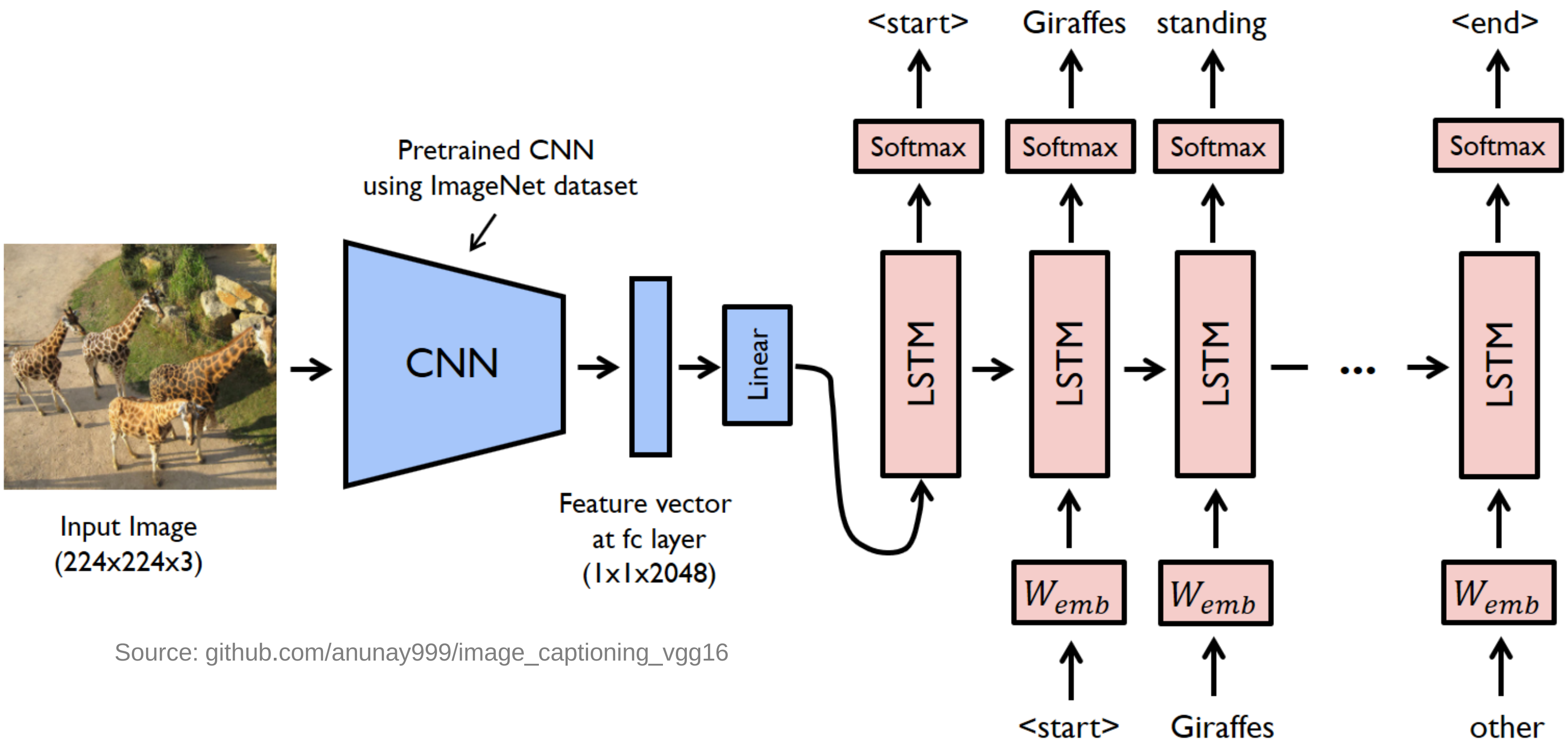
Applications of RNN

RNN for variable multi-slices input



Li et al. BMC Bioinformatics (2020)

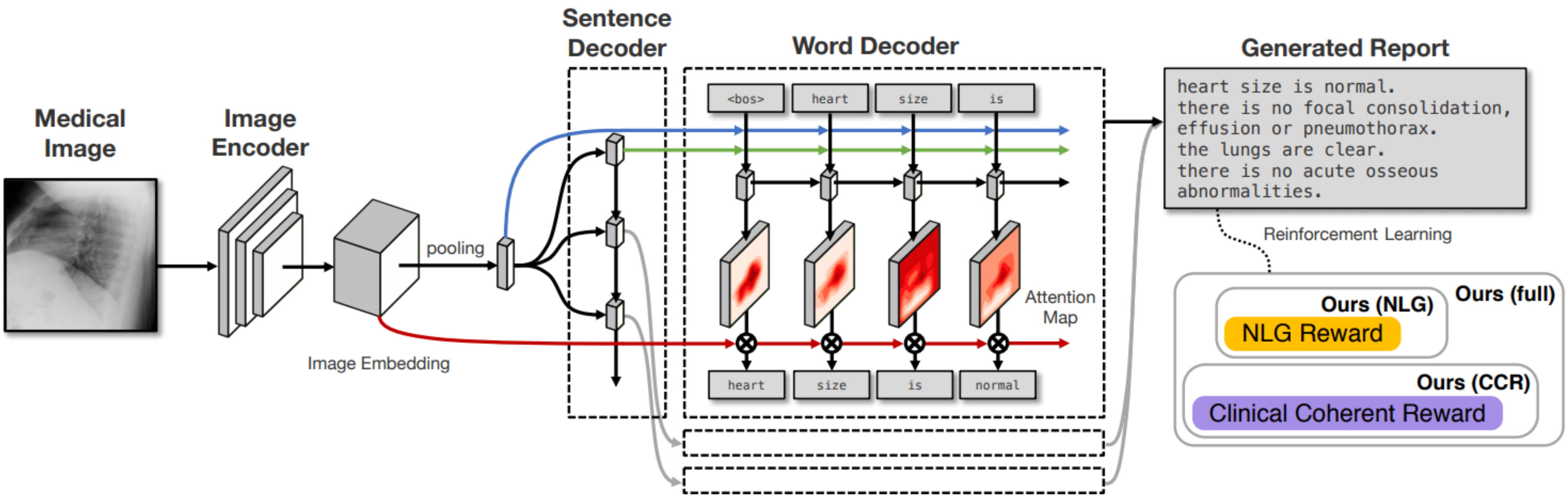
Image captioning



Source: github.com/anunay999/image_captioning_vgg16

- Embedding of input image is sent into an RNN to generate variable-length output sentence(s)
- Each output is a probability distribution over all words

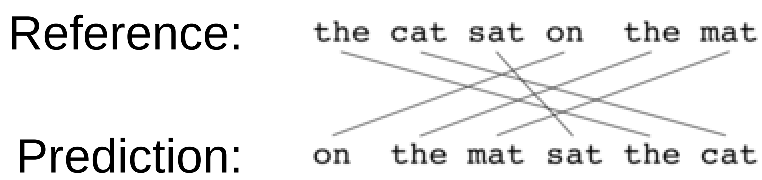
RNN for report generation



Liu et al. Clinically Accurate Chest X-Ray Report Generation (2019)

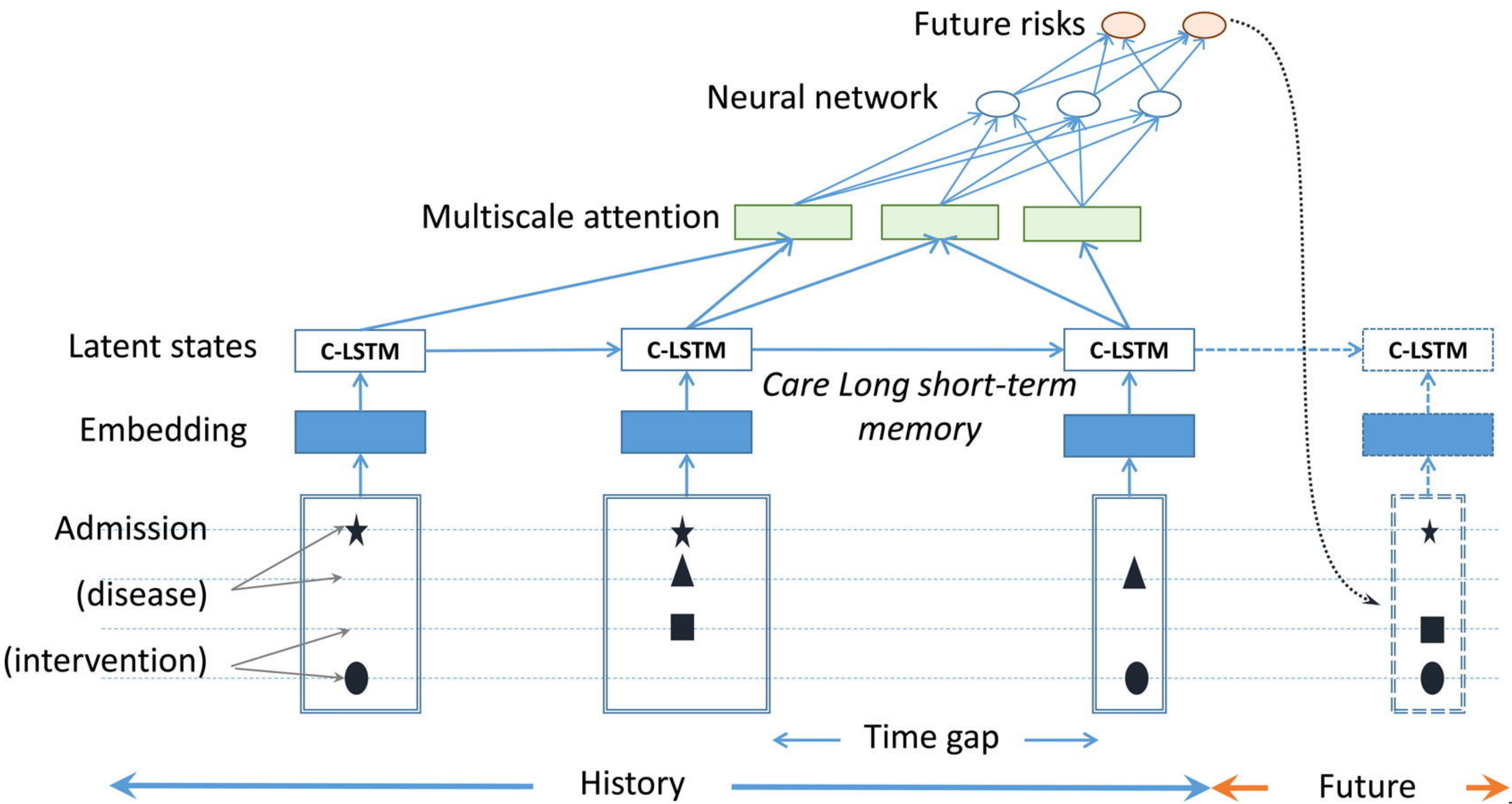
- Same principle as image captioning
- **Sentence decoder** generate “topic” + “stop token”
  - **Word decoder** then generate words that fill each sentence according to the predict “topic”
- 2D embedding of the input image is fed to the word decoder to relate each predicted word with location in input image

Metrics for sentence prediction



- BLEU (bilingual evaluation understudy)
  - Number of *n*-gram (groups of *n* words) that are common between the prediction and the reference
- METEOR (metric for evaluation of translation with explicit ordering)
  - $M = \frac{10\ \text{Precision} \cdot \text{Recall}}{\text{Recall} + 9\ \text{Precision}} \left(\frac{\text{Number of aligned word chunk}}{\text{Number of matched word}} \right)^3$
- ROUGE (recall-oriented understudy for gisting evaluation)
  - ROUGE-L = length of longest common subsequence between the prediction and the reference

RNN for EHR-based prediction



Pham et al. J of Biomed Informatics (2017)

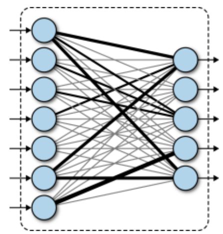
- Embed medical terms and feed time-series EHR to an RNN

Learn embedding without label

Embedding words

	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

Image from hackermoon.com



cat →	0.6	0.9	0.1	0.4	−0.7	−0.3	−0.2
kitten →	0.5	0.8	−0.1	0.2	−0.6	−0.5	−0.1
dog →	0.7	−0.1	0.4	0.3	−0.4	−0.1	−0.3
houses →	−0.8	−0.4	−0.5	0.1	−0.9	0.3	0.8
man →	0.6	−0.2	0.8	0.9	−0.1	−0.9	−0.7
woman →	0.7	0.3	0.9	−0.7	0.1	−0.5	−0.4
king →	0.5	−0.4	0.7	0.8	0.9	−0.7	−0.6
queen →	0.8	−0.1	0.8	−0.9	0.8	−0.5	−0.9

Image from medium.com

- Word encoder = a fully connected neural network
  - Input = one-hot encoding of words
  - Output = a continuous-value embedding
  - Used as encoder for a larger network (transfer learning)

Word embedding based on context

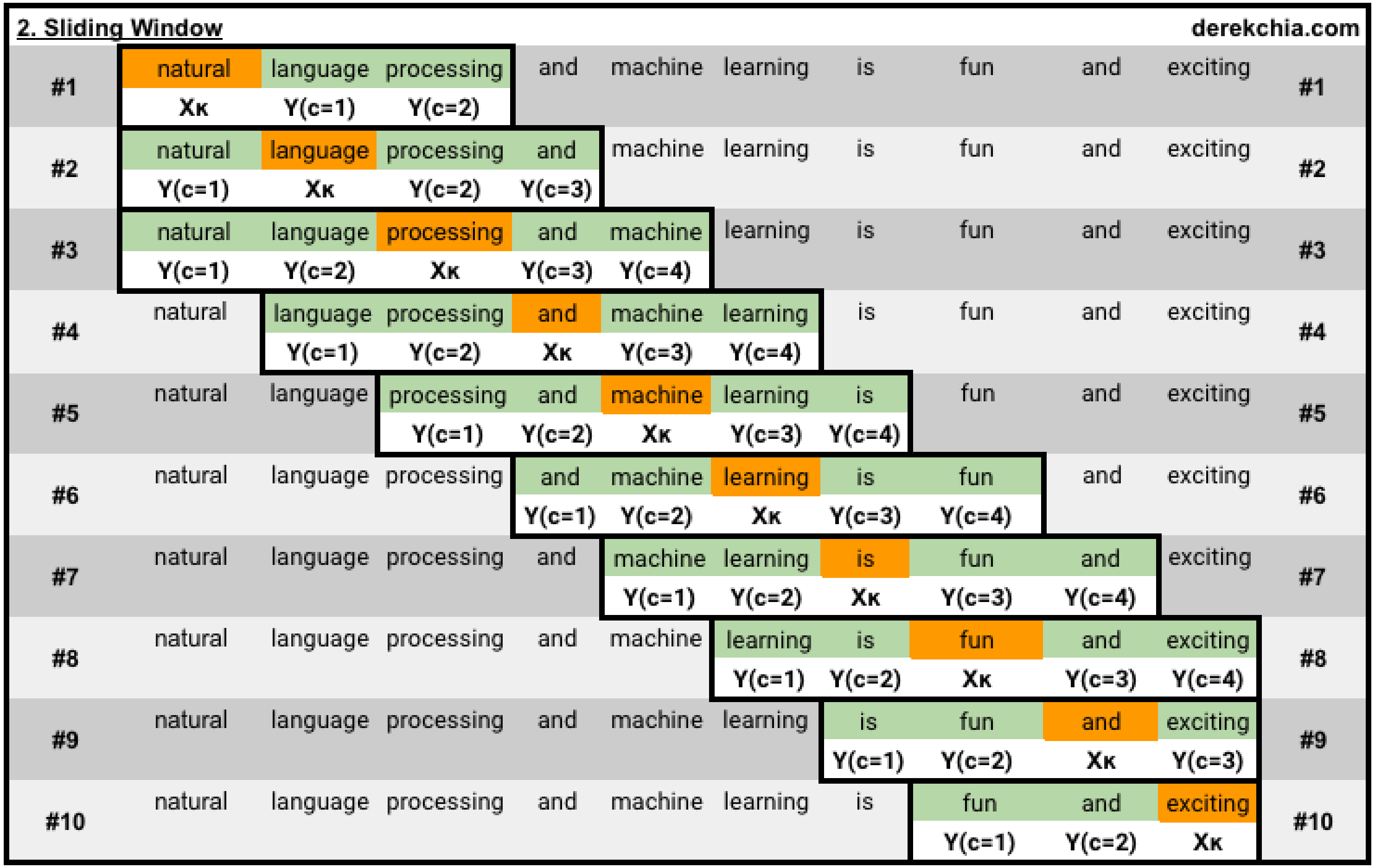


Image from towardsdatascience.com/an-implementation-guide-to-word2vec-using-numpy-and-google-sheets-13445eebd281

- Embedding of adjacent words should be similar
- Embedding should be able to predict context relationship

Skip-gram model

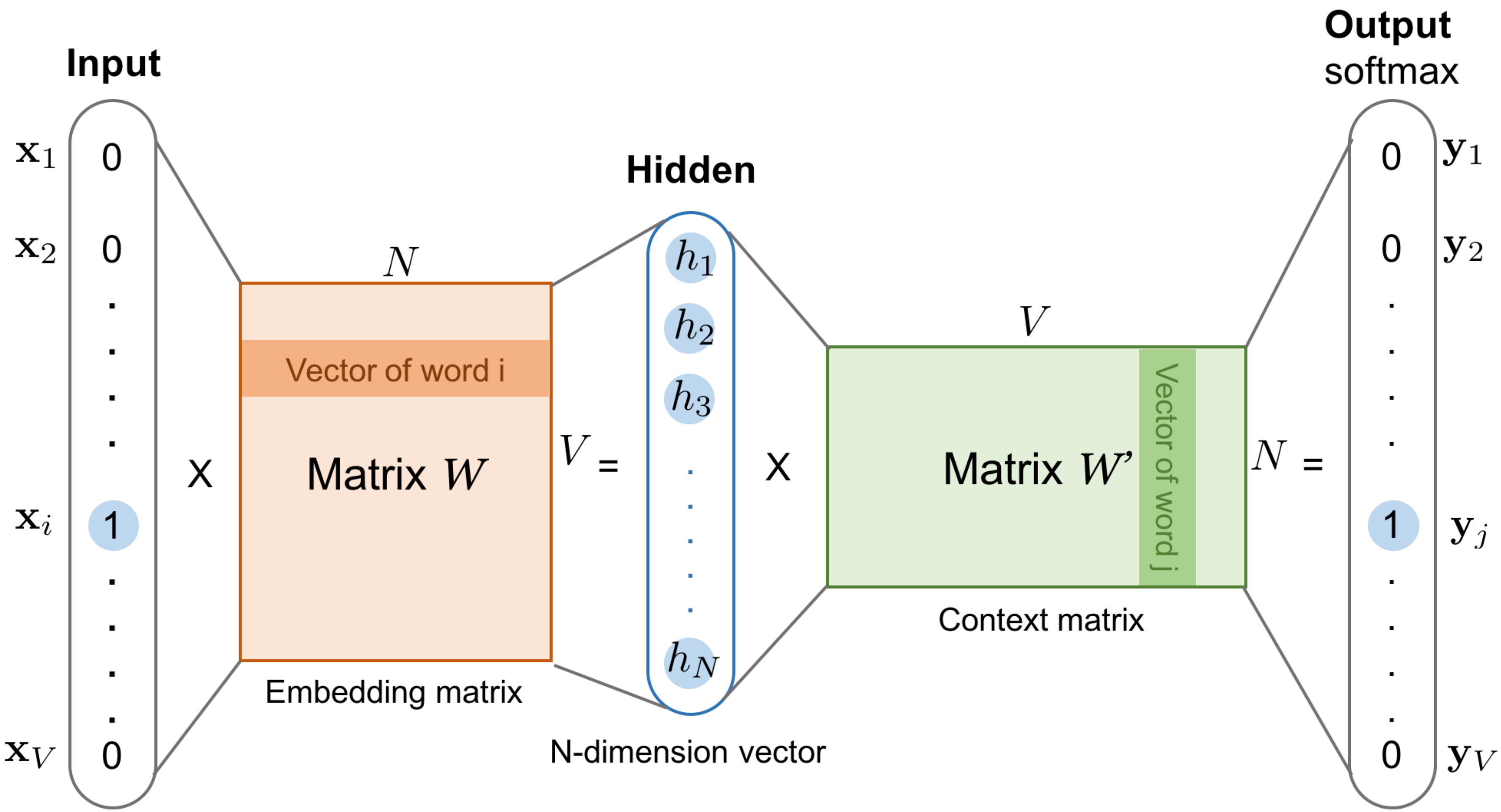
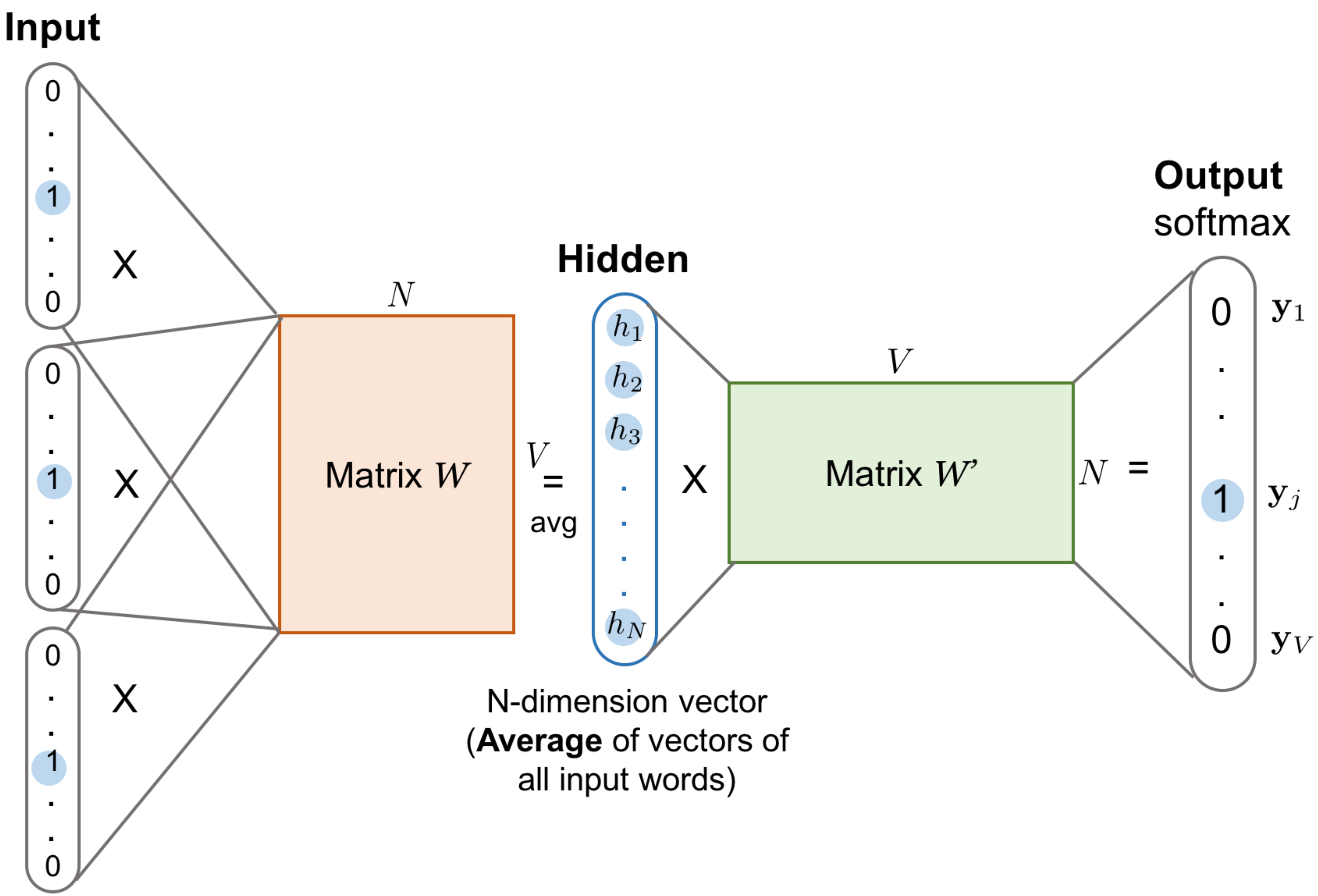


Image from towardsdatascience.com/an-implementation-guide-to-word2vec-using-numpy-and-google-sheets-13445eebd281

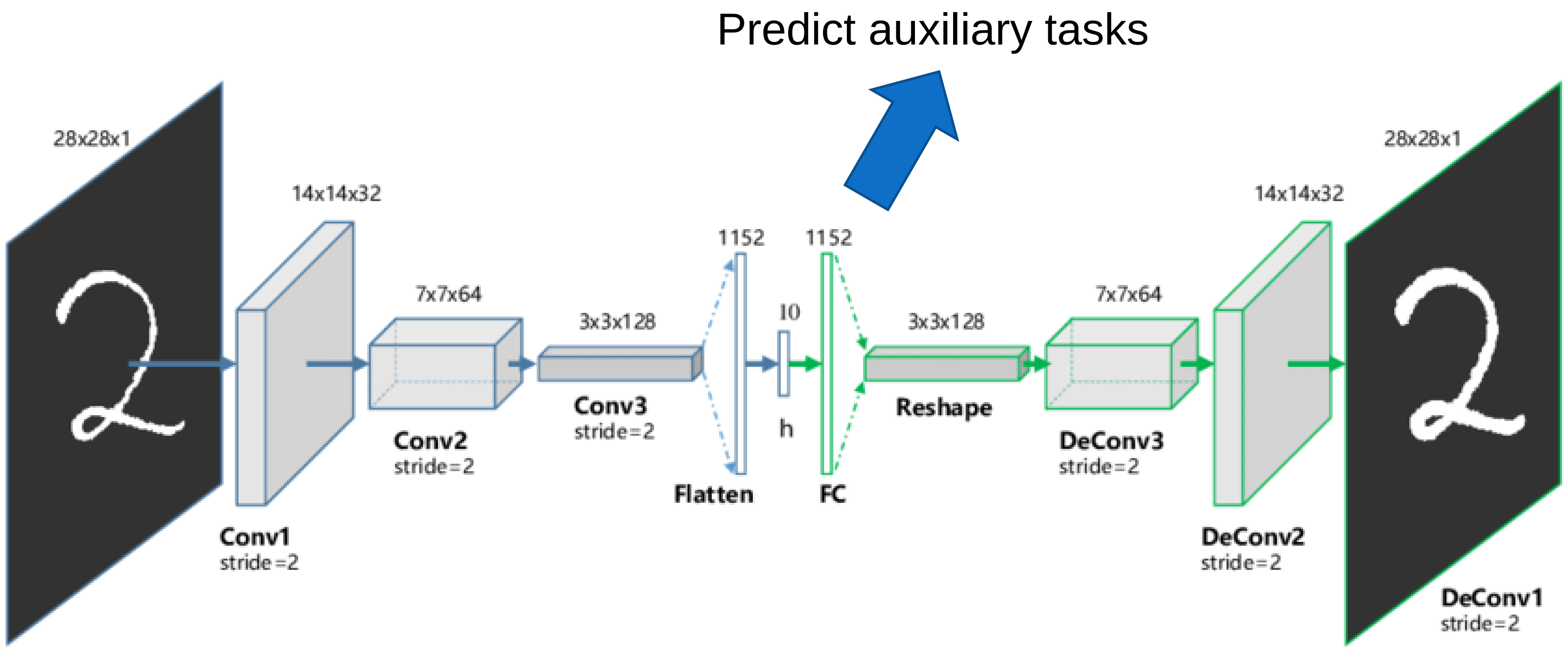
- Predict context words from an input word
  - Training data = pair of context words

Continuous bag-of-words model



- Predict center word from average embeddings of context words

Autoencoder



Guo et al. International Conference on Neural Information Processing (2017)

- Train the model to predict input data themselves
- But squeeze the embedding dimension (output of the flatten layer above) to prevent the model from just "remembering"
  - This is called "bottleneck"
- This also reproduced all noises and errors

Denoising autoencoder

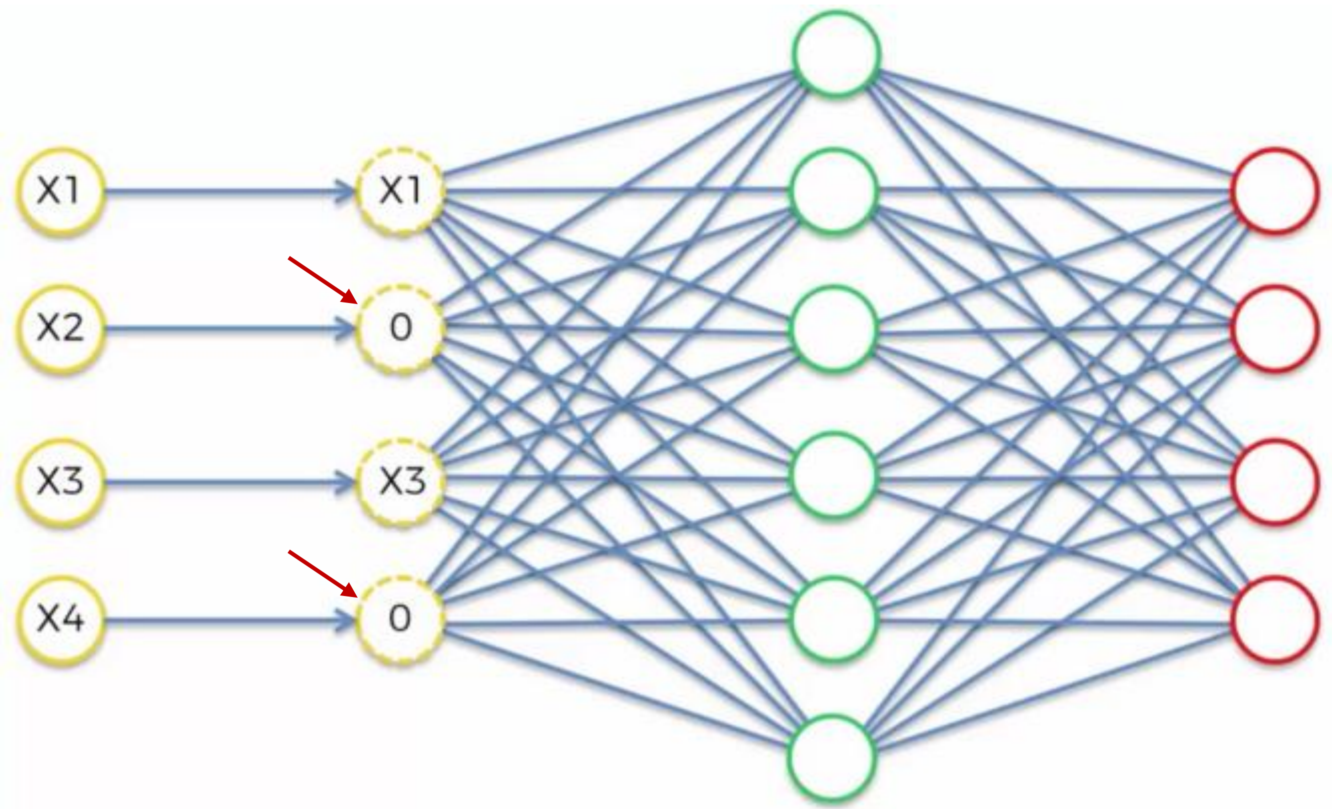
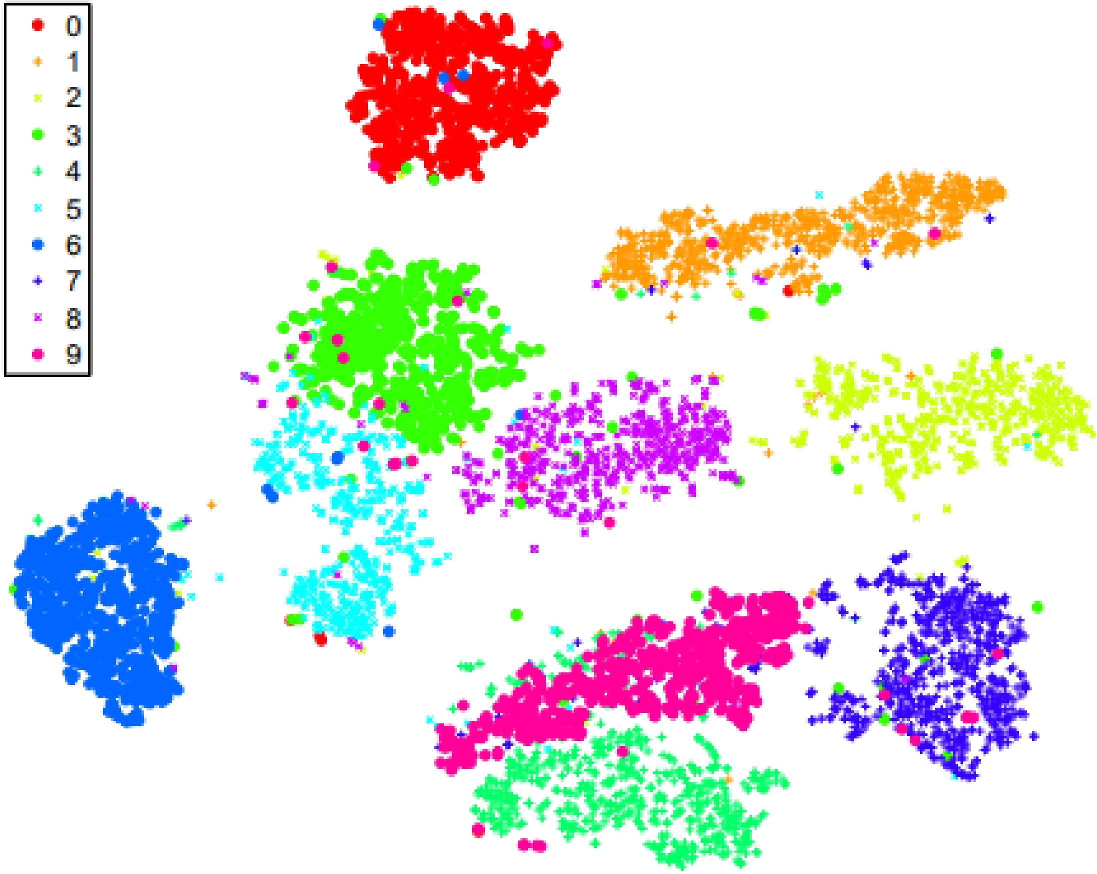


Image from towardsdatascience.com/denoising-autoencoders-explained-dbb82467fc2

- Introduce noises into the input before feeding into the model
- But calculate loss based on original input
  - Prevent model from overfitting to noises and errors in input
  - Similar to Dropout

Good embedding



Maaten, L. and Hinton, G. J of Machine Learning Research 9:2579-2605 (2008)

- Embedding of data from the same class should be nearby
- Embedding of data from “similar” classes should be nearby

Variational autoencoder (VAE)

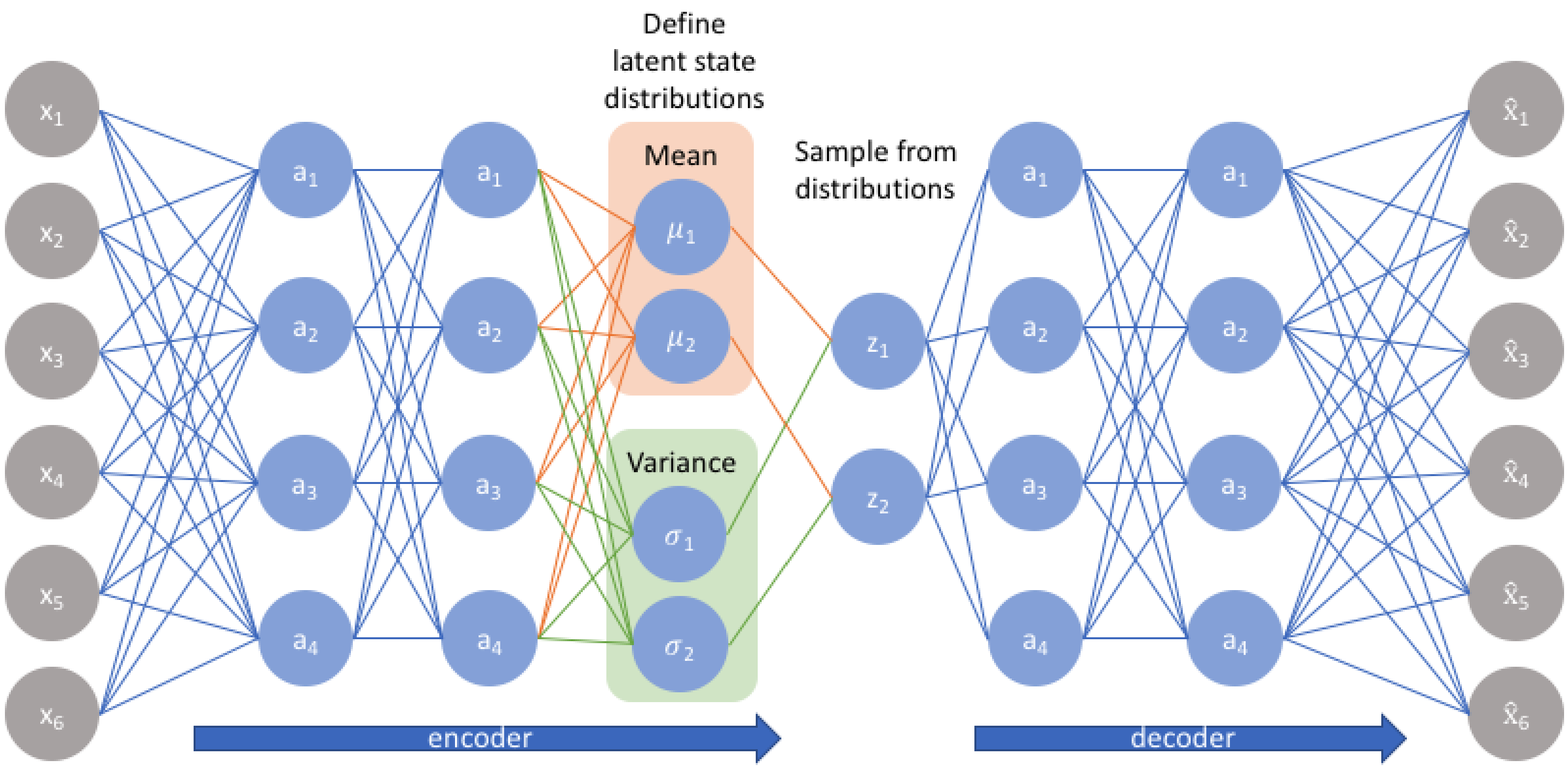


Image from www.jeremyjordan.me/variational-autoencoders/

- Embed the mean and variance
- Sample from Normal distribution for the decoder
  - $z_i \sim \text{Normal}(\mu_i, \sigma_i) = \mu_i + \sigma_i \times \text{Normal}(0, 1)$
  - Backpropagate through $z_i \rightarrow (\mu_i, \sigma_i)$

Loss function for VAE

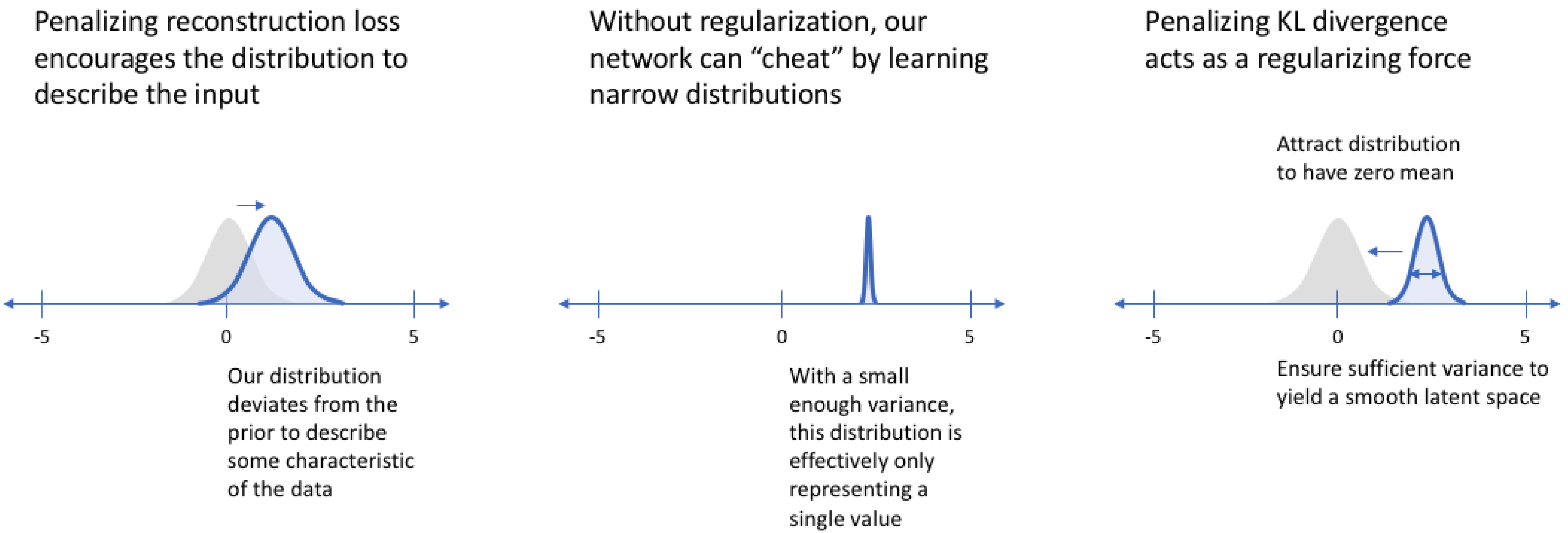


Image from www.jeremyjordan.me/variational-autoencoders/

- "Neural network always tries to cheat"
- Reconstruction loss = $L(x, \hat{x})$
  - The model should be able to reconstruct the input data
- KL divergence loss = $\mathrm{KL}(\mathrm{p}(\mathrm{z} \mid \mu, \sigma) \,||\, \mathrm{Normal}(0, 1))$
  - Force the model to sample from normal distribution

Object detection network

R-CNN

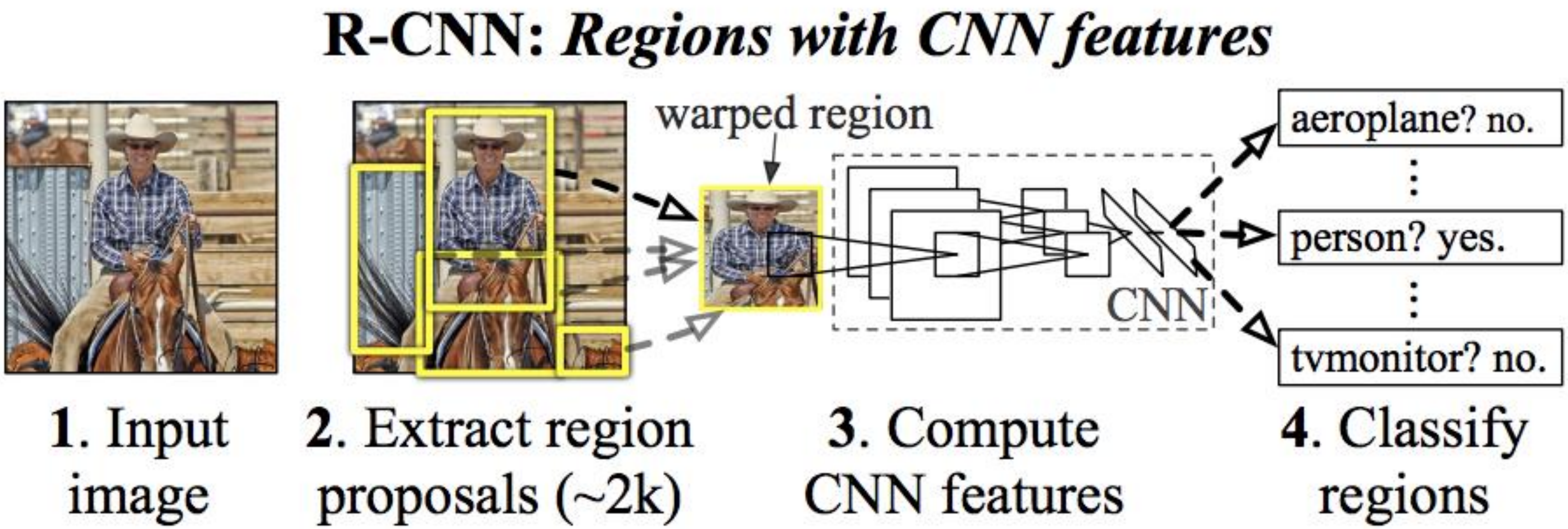


Image from towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

- A fixed region proposal algorithm
- Feature extraction for each region using CNN
- Classification with SVM
- Also predict offset for adjusting the bounding box location
  - Proposed region might not contain the whole object

Fast R-CNN

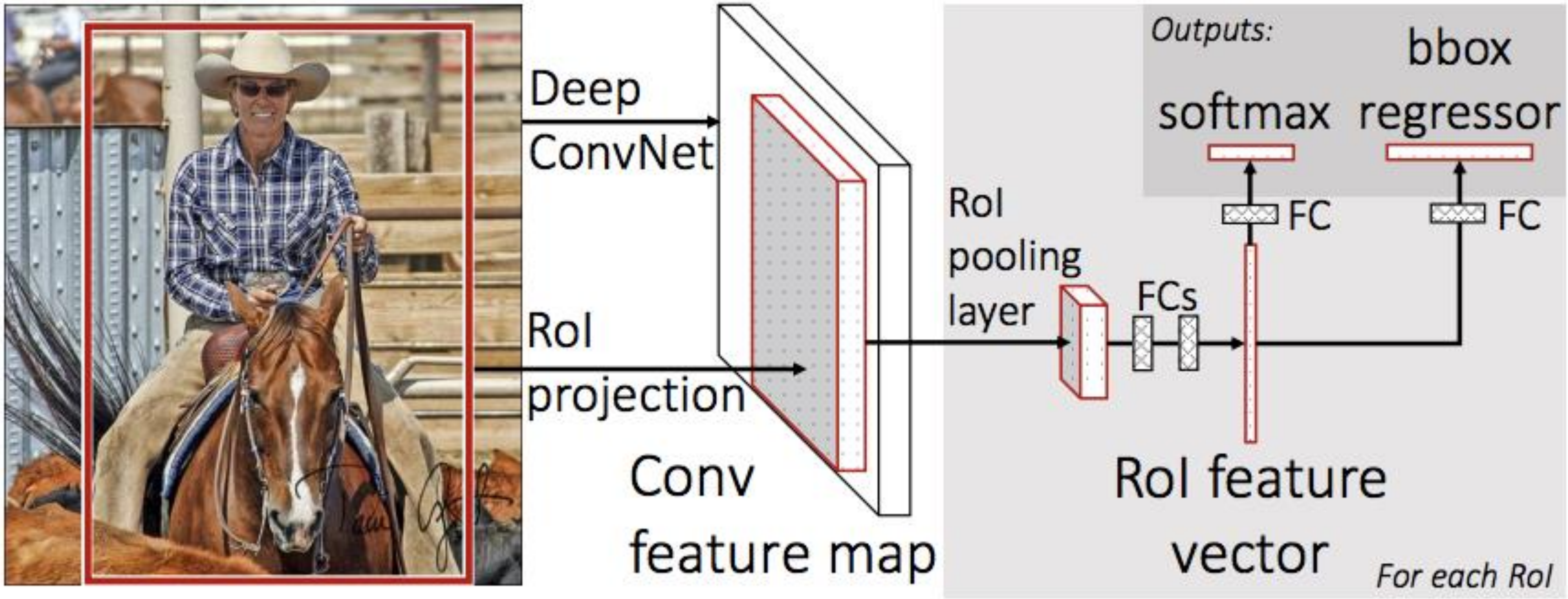


Image from towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

- Input the whole image through CNN
- Replace global pooling with local pooling
  - Correspond to each proposed region

Faster R-CNN

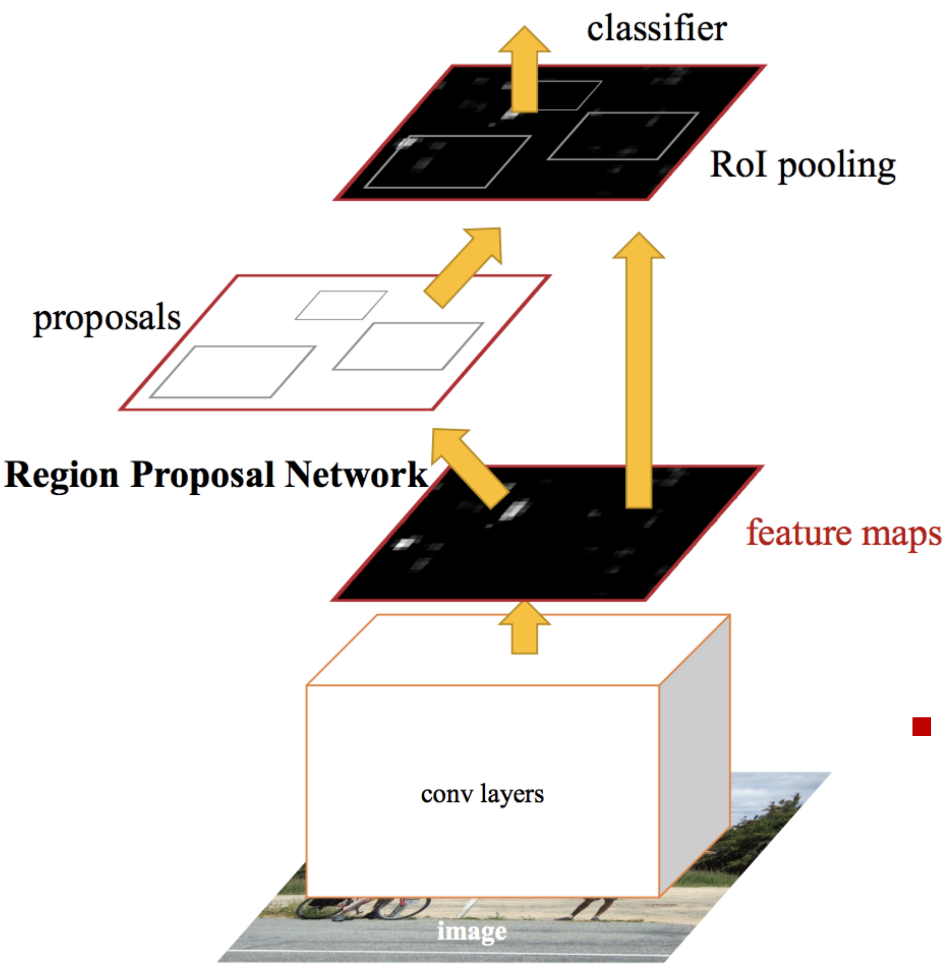
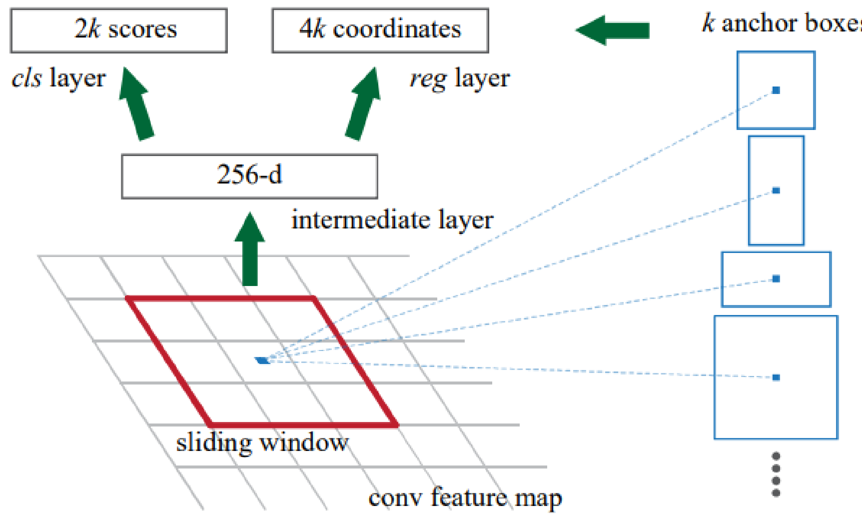
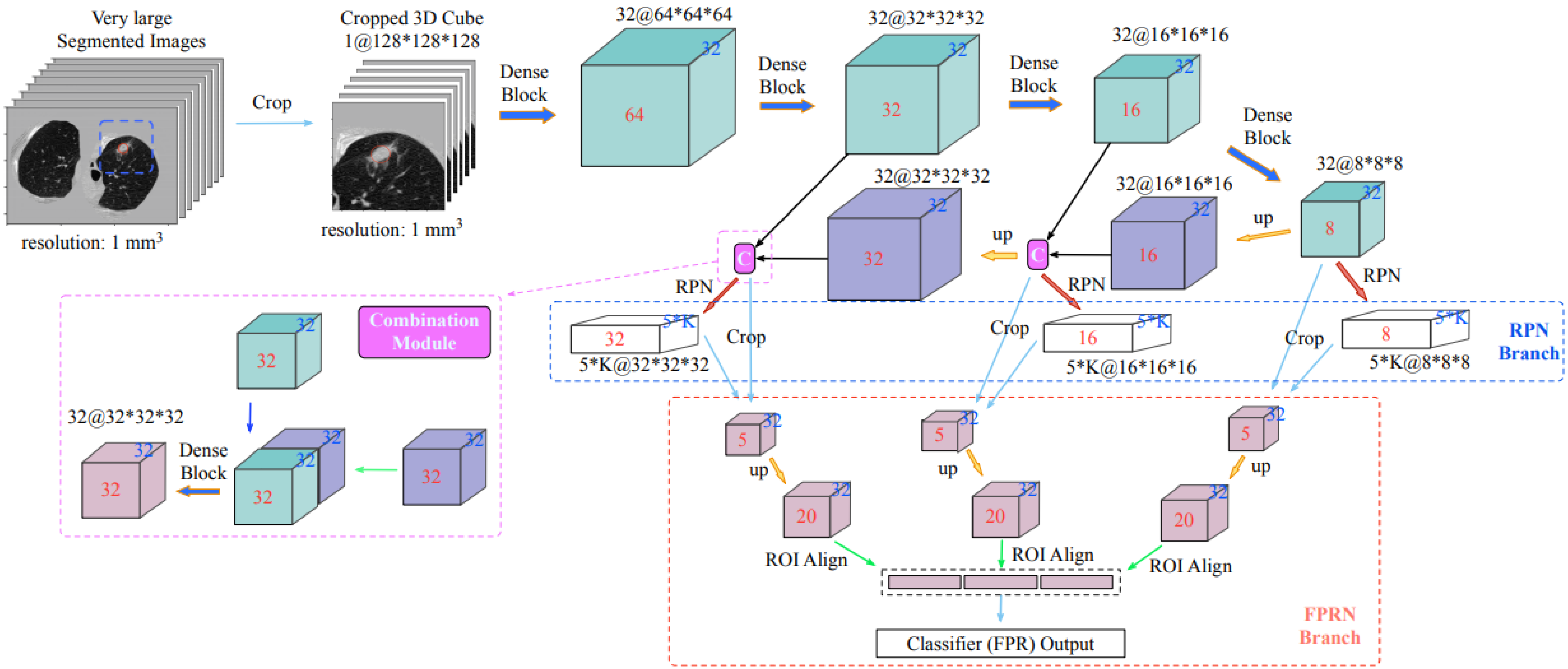


Image from towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e



- Region proposal network is a CNN which predicts *k* bounding boxes as well as their “objectness” scores
- 25x faster than Fast R-CNN

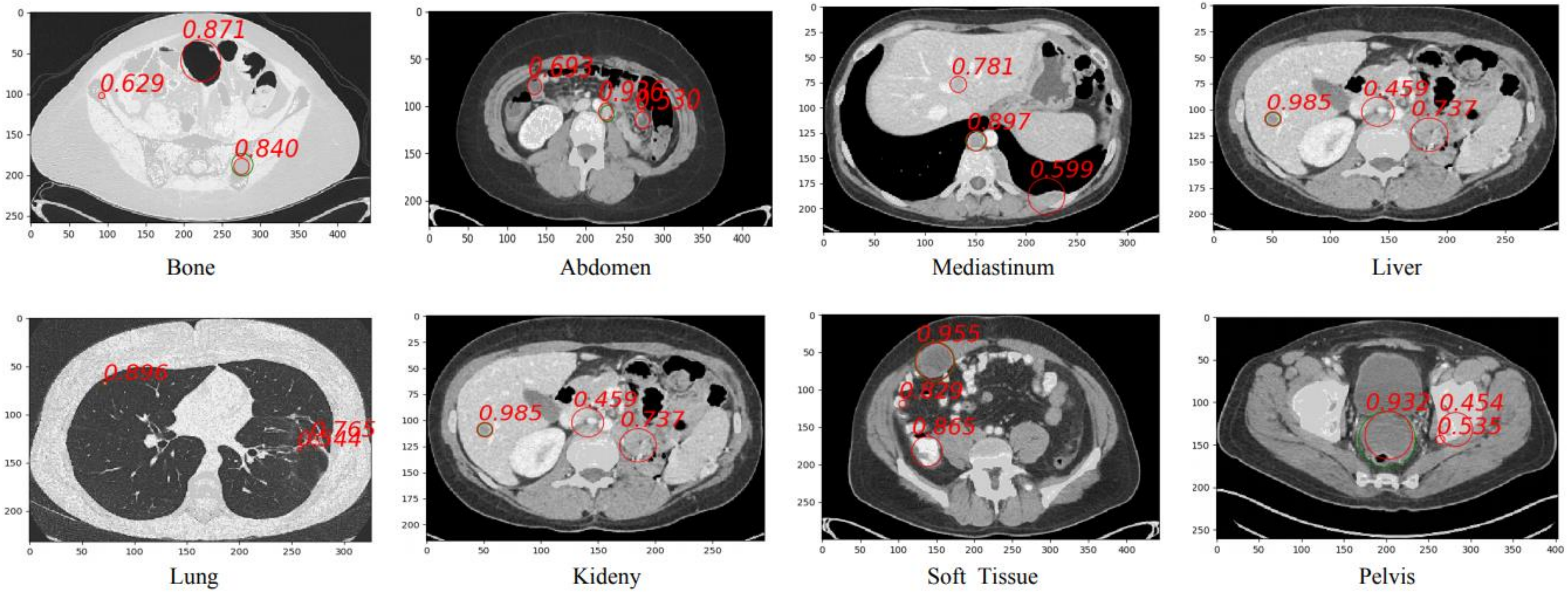
Faster R-CNN on CT images



Zhang et al. “3D Aggregated Faster R-CNN for General Lesion Detection” arxiv.org/pdf/2001.11071

- U-Net + Region Proposal Network
- Propose regions at multiple scales
- Combine data from all scales for classification

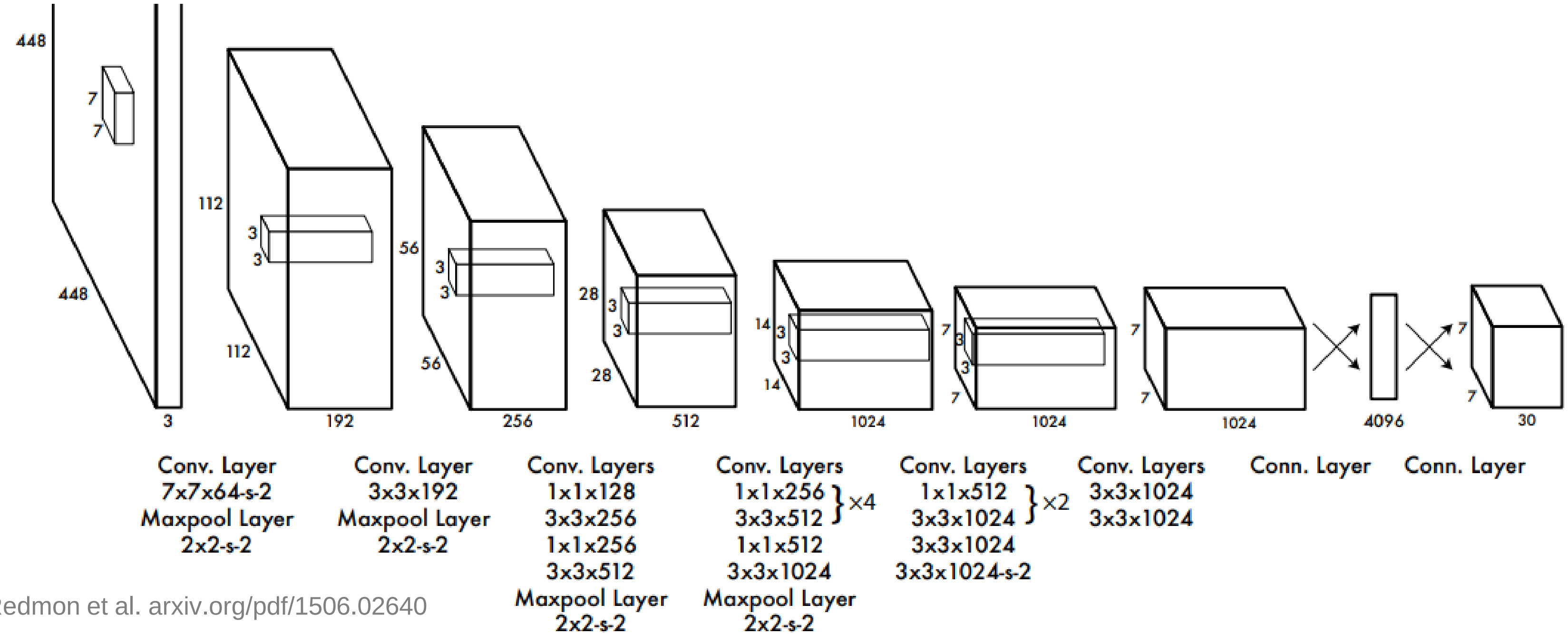
Faster R-CNN on CT images



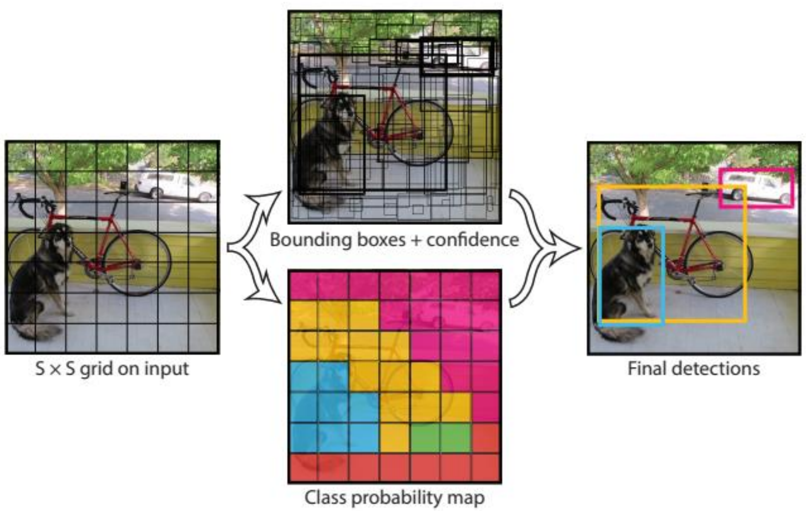
Zhang et al. “3D Aggregated Faster R-CNN for General Lesion Detection” arxiv.org/pdf/2001.11071

- Deep Lesion dataset (NIH releases 32,000 CT images)
- LUNA16 challenge (Annotated nodules in 888 CT images)

YOLO – You only look once



Redmon et al. arxiv.org/pdf/1506.02640



- Predict several SxS matrices
  - Bounding box locations
  - Bounding box confidence scores
  - Class confidence scores
- Single evaluation in CNN
- 45-155 FPS

Any question?