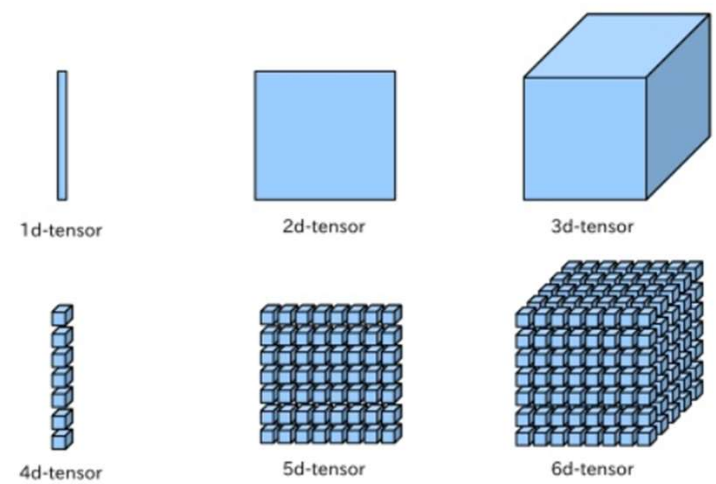
# Week 2 – Neural Network

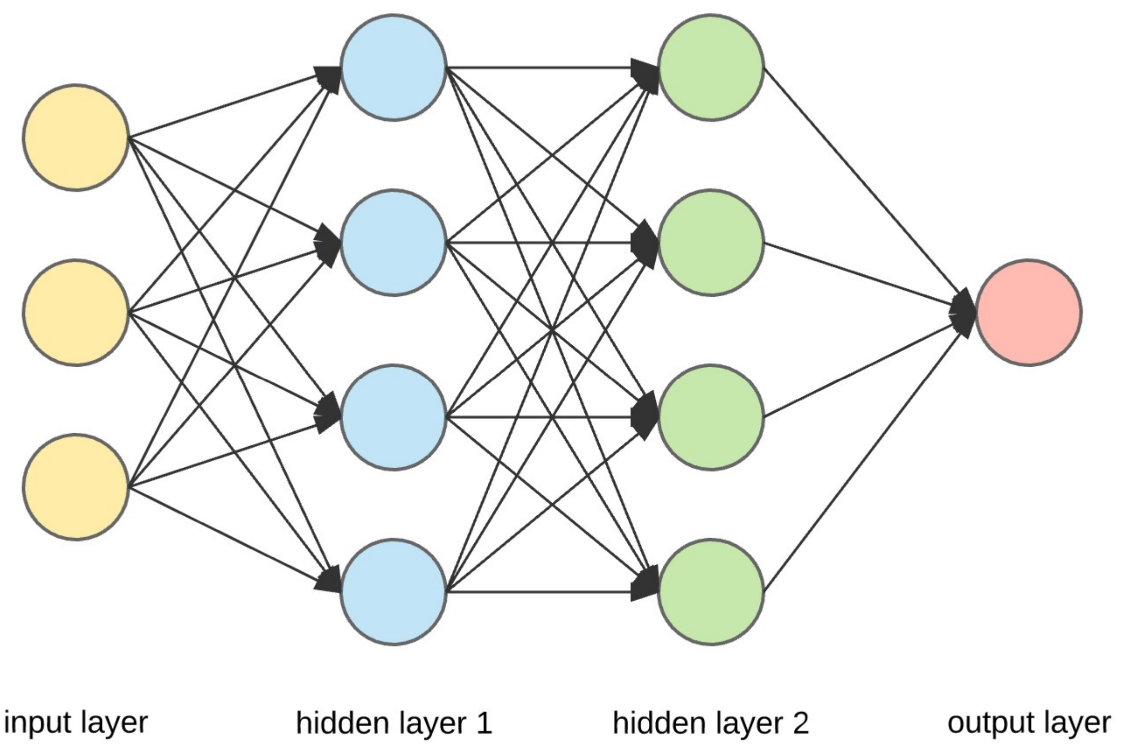
EGCO467 Natural Language and Speech Processing

# Tensor Shapes

- Convolution: 4D (N, w, h, c)
- Dense: 2D (N, features)
- LSTM: 3D (N, time, feature)

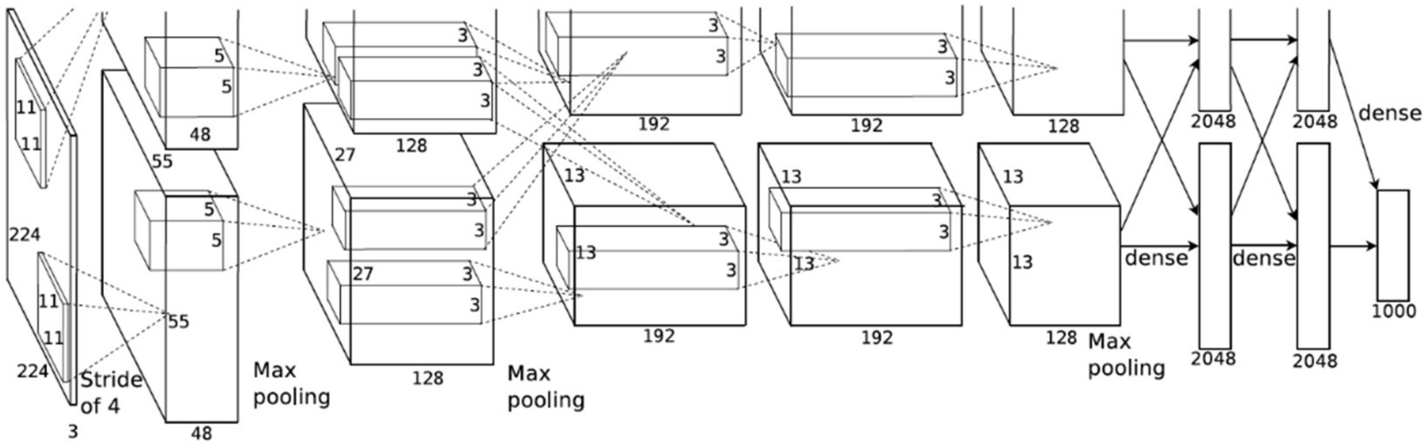


# FNN




credit: https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

# CNN




Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton.
"Imagenet classification with deep convolutional neural networks."
*Advances in neural information processing systems*. 2012.

# NN training

$$min_\theta = J(f_\theta(X), Y)$$

$X$ = training inputs

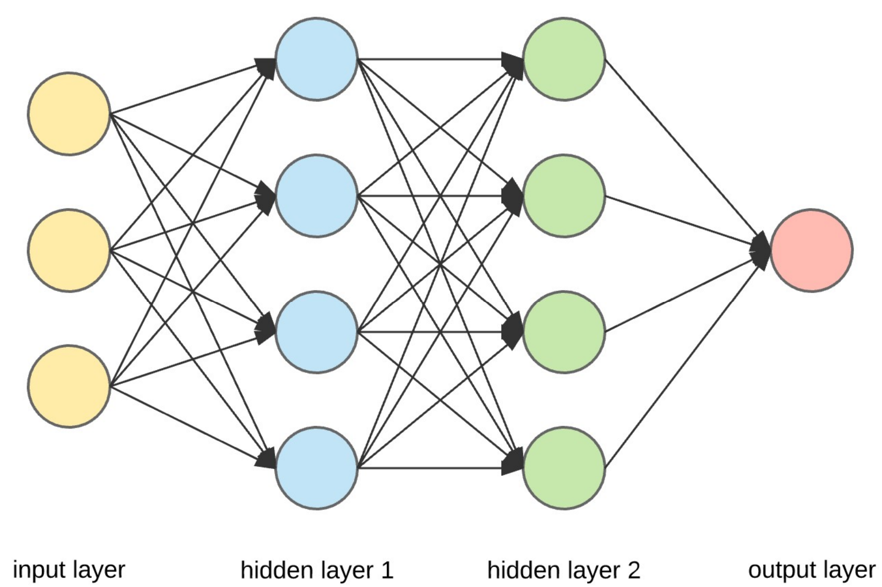
$Y$ = training outputs

$f_\theta$ = NN

$J$ = objective (loss) function: N->1 function

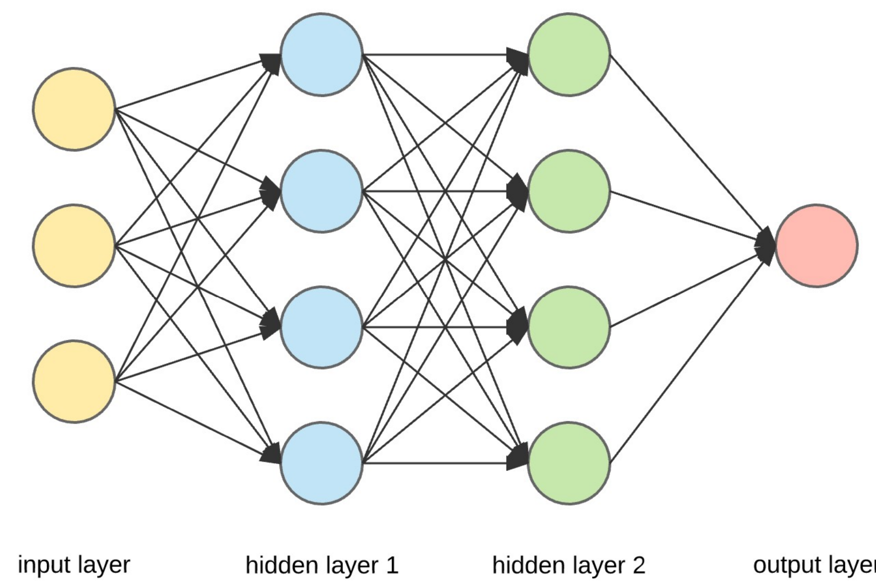
$\theta$ = weights

try to decrease $J$ by adjusting $\theta$



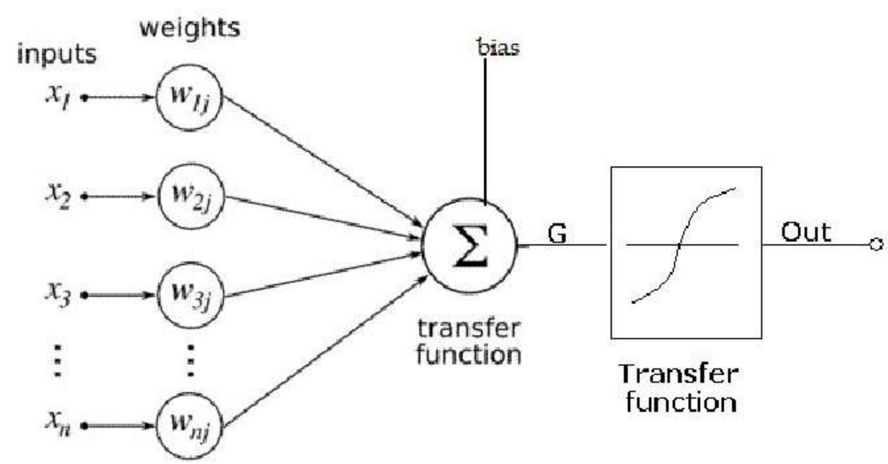
# Dense (linear) layer

- $y = Ax + b$
- $(M \times 1) = (M \times N)(N \times 1) + (M \times 1)$



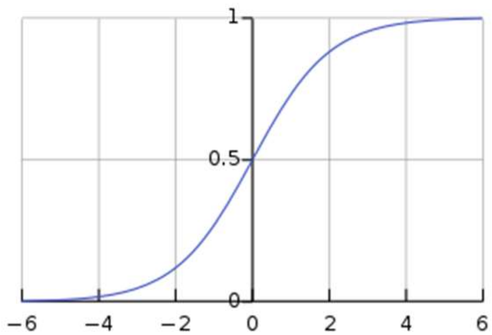
# Nonlinearity

- Without nonlinearity:
- $(A_3(A_2(A_1x))) = A_Tx$
  where $A_T = A_3A_2A_1$

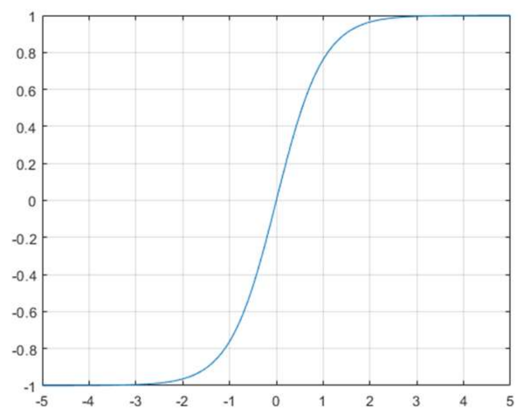



credit: https://www.researchgate.net/figure/Artificial-Neuron-model_fig4_277774116

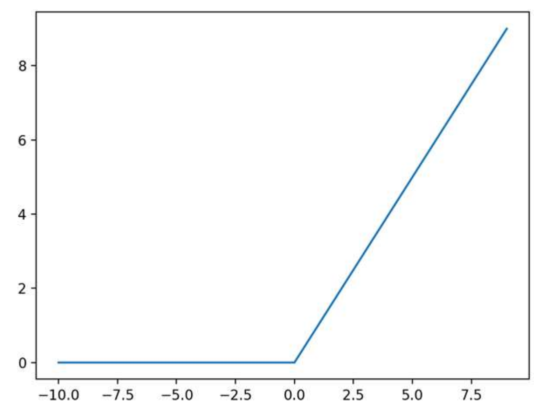
# Activation function




sigmoid



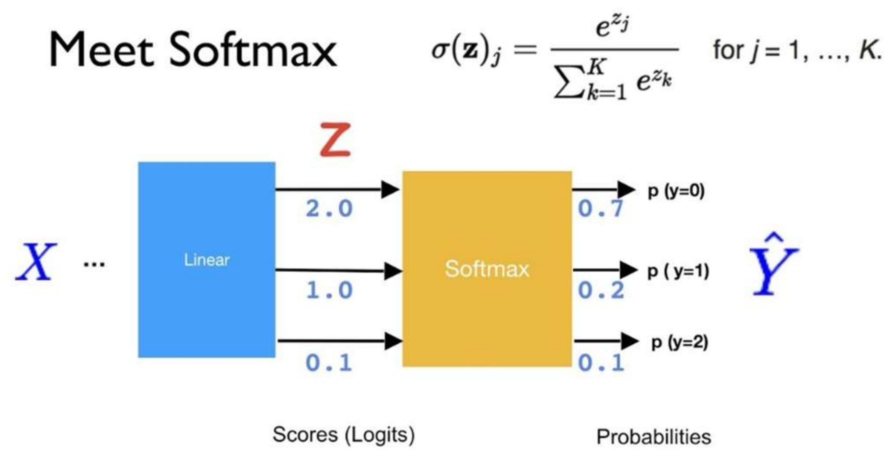

tanh




relu

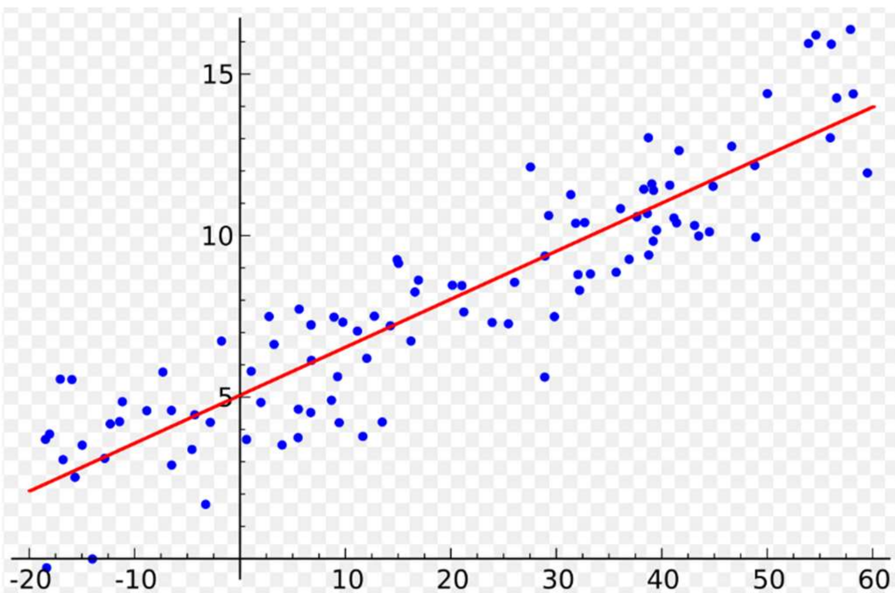
# Softmax

- Map N real numbers (-inf, inf) -> (0,1)
- all the outputs sum to 1
- for classification

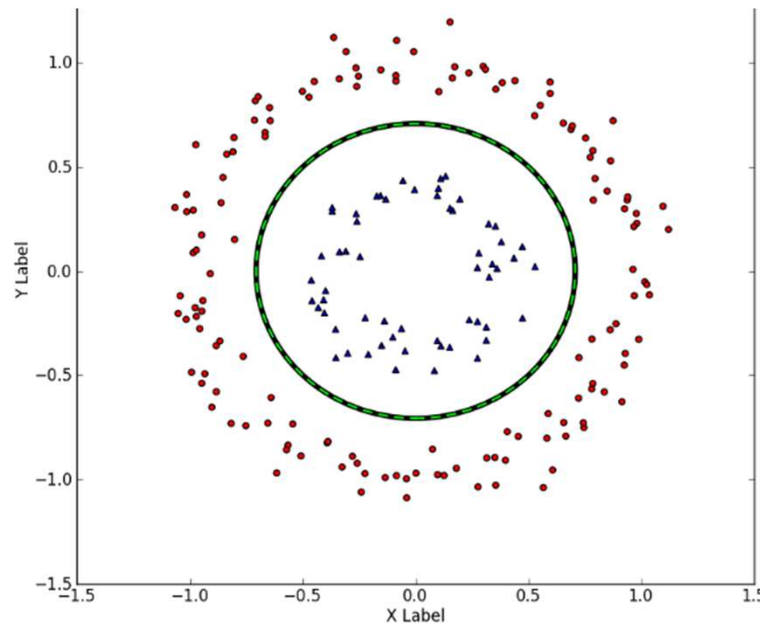



credit: https://www.youtube.com/watch?v=lvNdl7yg4Pg

# Regression vs. classification

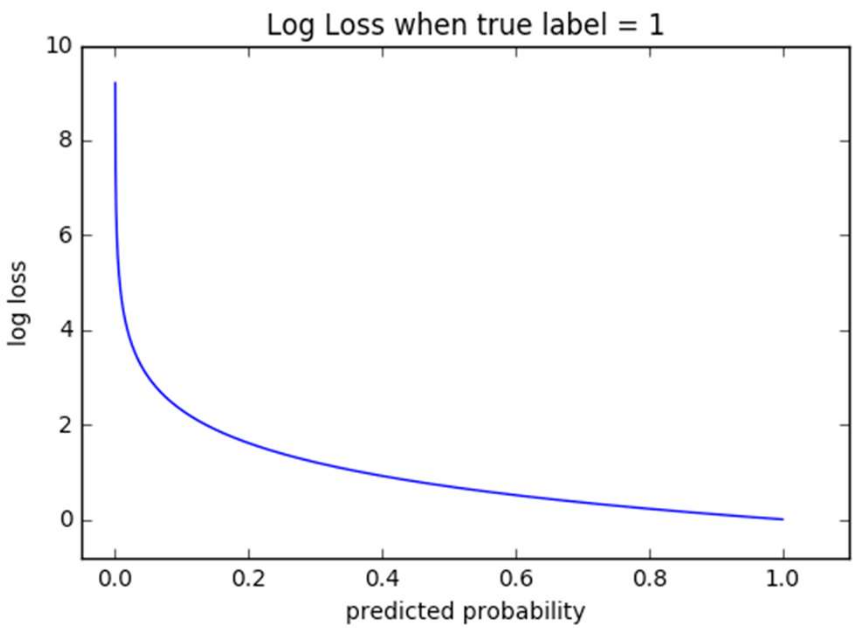



https://en.wikipedia.org/wiki/Linear_regression




https://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html

# Loss function for classification




Cross-entropy

In binary classification, where the number of classes $M$ equals 2, cross-entropy can be calculated as:

$$-(y\log(p) + (1-y)\log(1-p))$$

If $M > 2$ (i.e. multiclass classification), we calculate a separate loss for each class label per observation and sum the result.

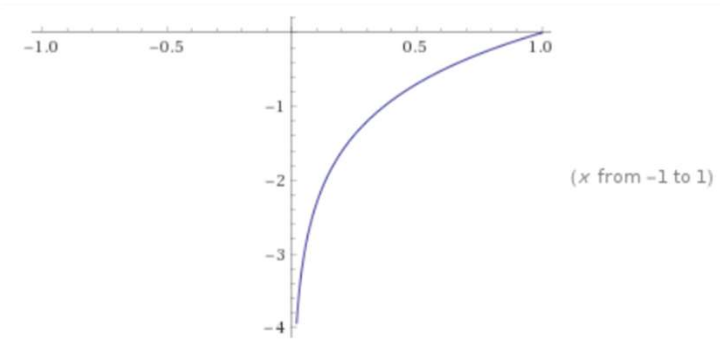
$$-\sum_{c=1}^{M} y_{o,c}\log(p_{o,c})$$

output of softmax

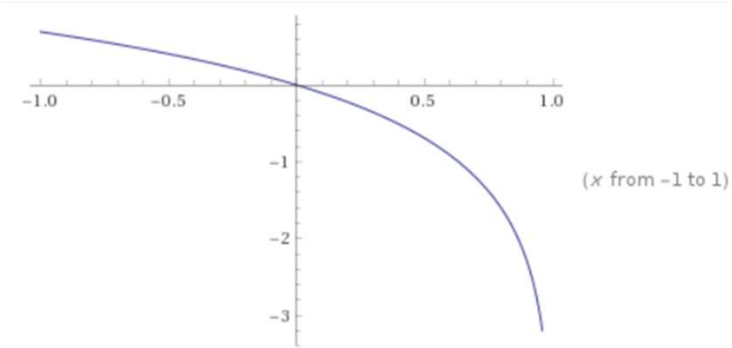
https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html#cross-entropy

# Loss function for classification

log(p)




log(1-p)

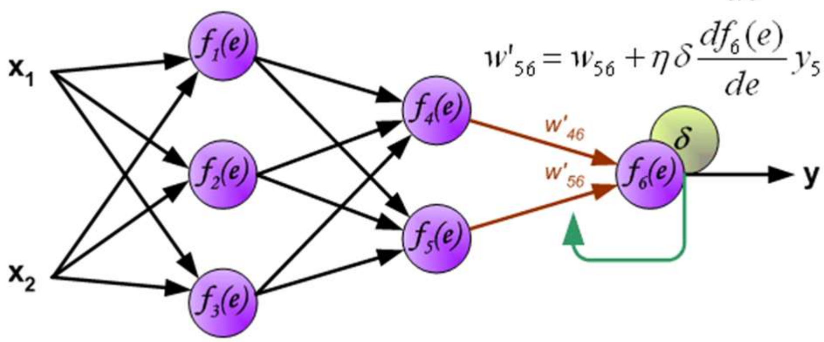



$$-(y\log(p) + (1-y)\log(1-p))$$

# Backpropagation (gradient descent)

$$w'_{46} = w_{46} + \eta\,\delta \frac{df_6(e)}{de} y_4$$

$$w'_{56} = w_{56} + \eta\,\delta \frac{df_6(e)}{de} y_5$$




The error between the actual output and the "true" output is $\delta = y_{\mathrm{true}} - y = y_{\mathrm{true}} - f_6(e)$.

Let $e_i$ to mean the activation of node $i$. We define the loss function to be the instantaneous mean square error

$$L = \frac{1}{2}\delta^2$$

$$\frac{\partial L}{\partial w_{46}} = \frac{\partial L}{\partial \delta}\frac{\partial \delta}{\partial f_6(e_6)}\frac{\partial f_6(e_6)}{\partial e_6}\frac{\partial e_6}{\partial w_{46}}$$

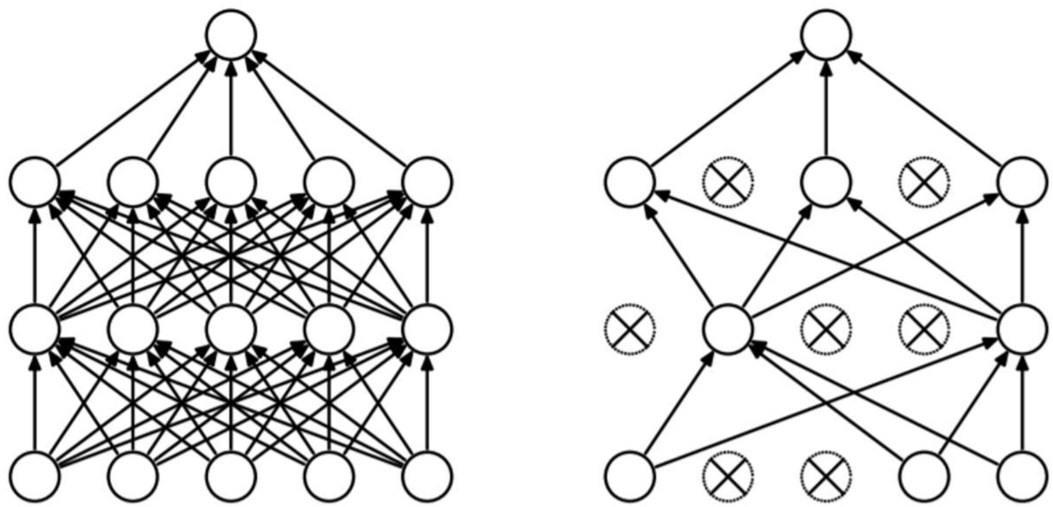
Since $e_6 = w_{46}y_4 + w_{56}y_5$, we have

$$\frac{\partial L}{\partial w_{46}} = (-1)\delta f_6'(e_6) y_4$$

and similarly for $w_{56}$.

http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

# Dropout layer

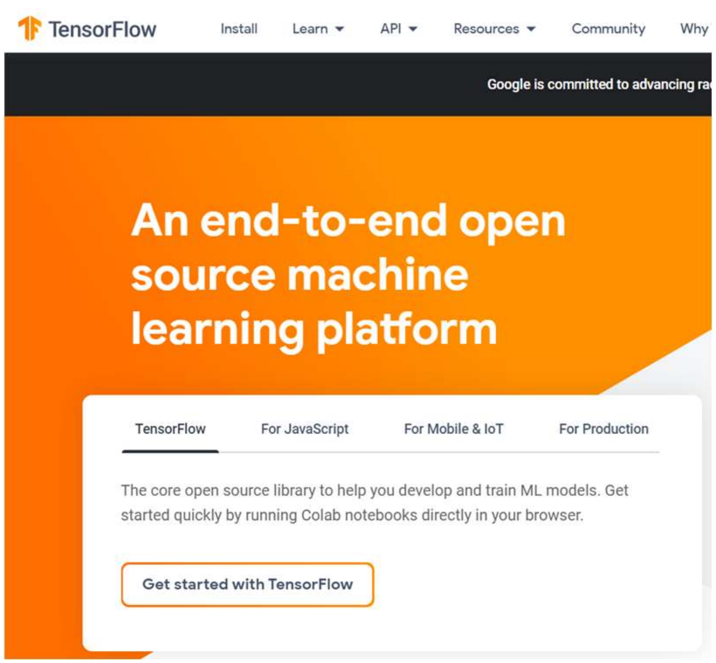


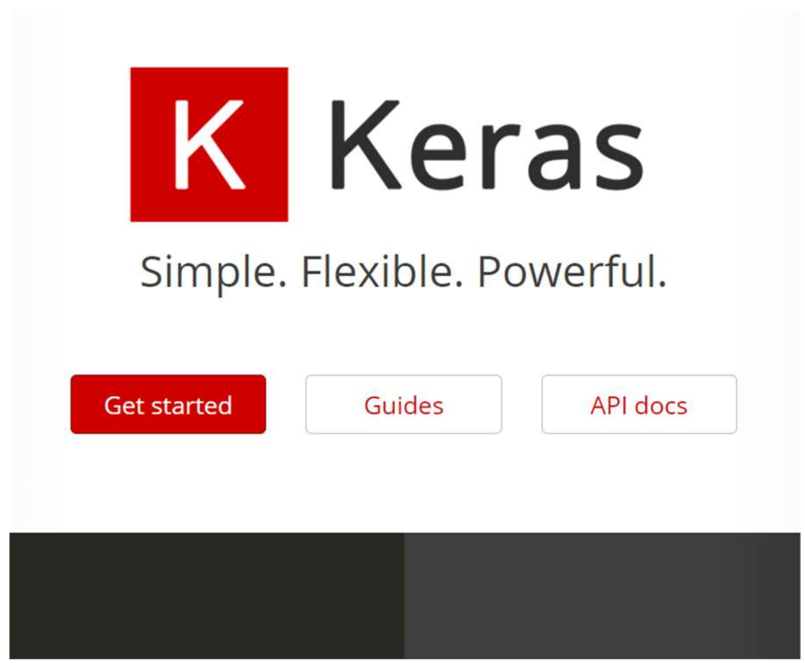
(a) Standard Neural Net

(b) After applying dropout.

Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." The journal of machine learning research 15.1 (2014): 1929-1958.

# DL Libraries



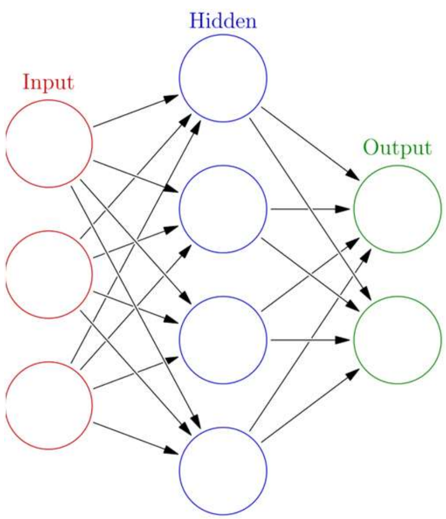
# Example 1 - MNIST

- https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/quickstart/beginner.ipynb

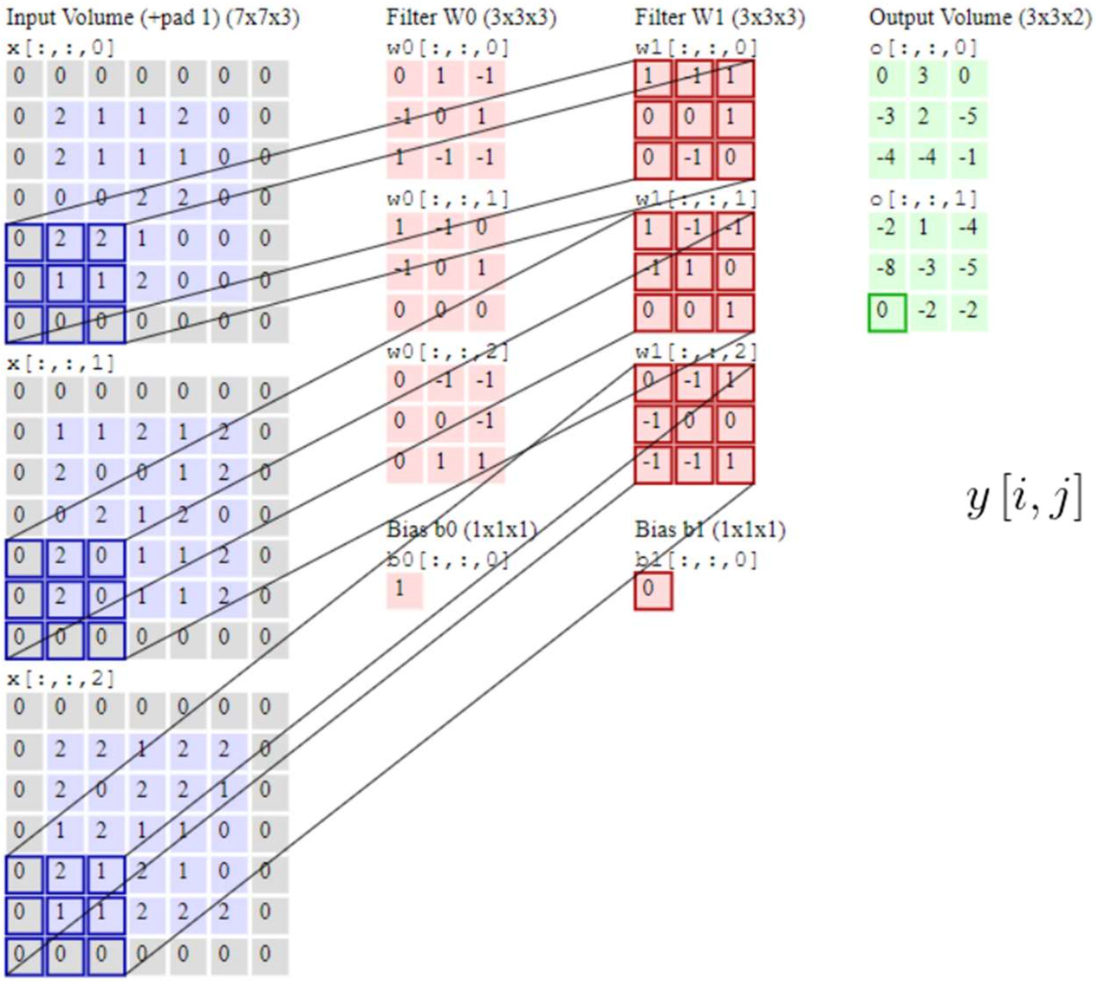
# CNN

# Convolutional Neural Network (CNN)

- Dense has to many weights



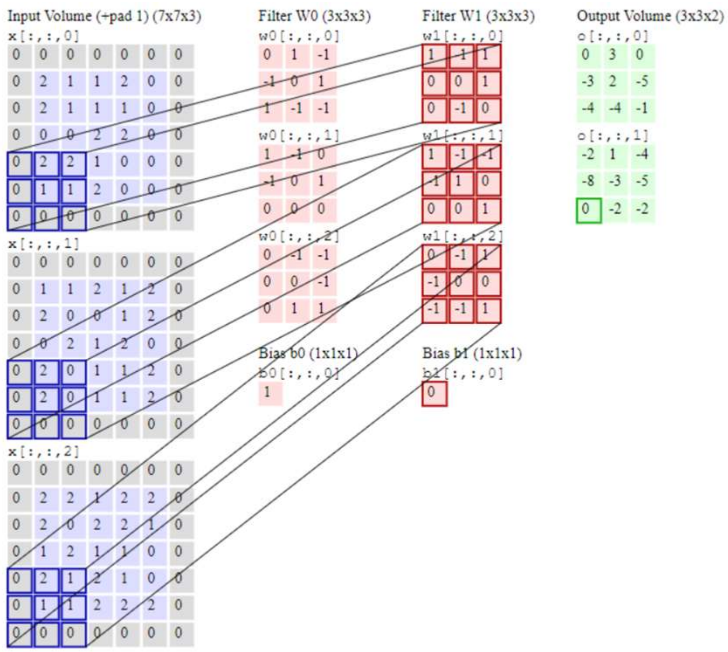

- Input = 1M pixel image
- Dense with 100 units = 100M weights
- 2D information is destroyed by reshape




$$y[i,j] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} h[m,n] \cdot x[i-m, j-n]$$

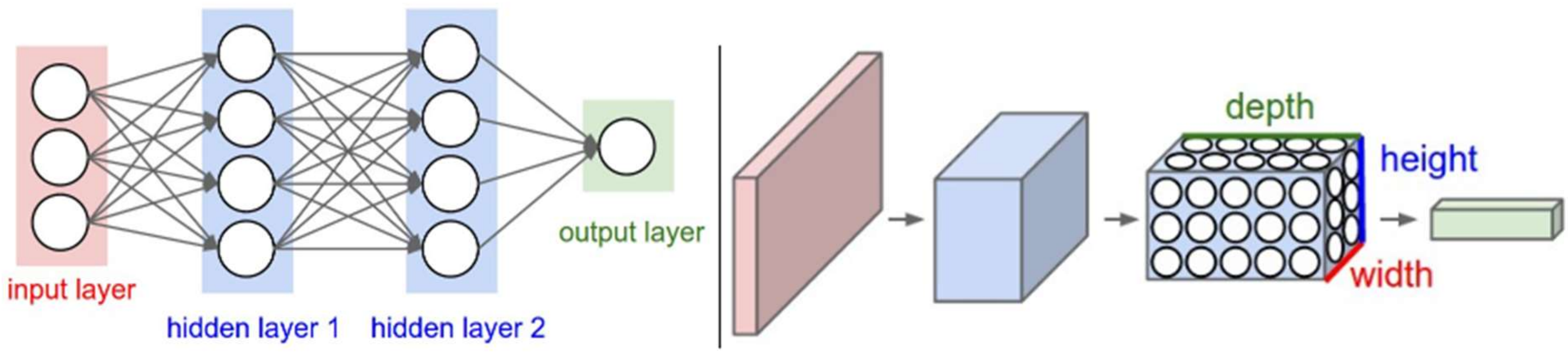
http://cs231n.github.io/convolutional-networks/

# Convolutional Layer



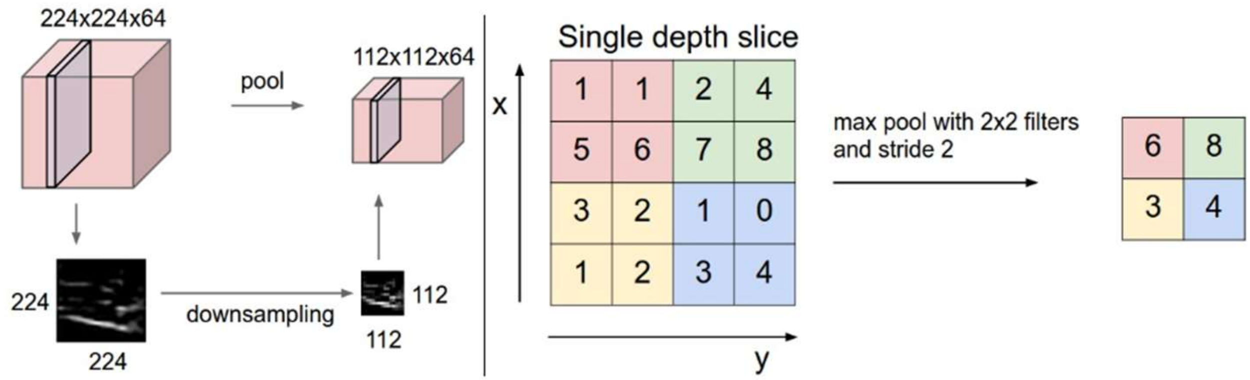

```
tf.keras.layers.Conv2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding="valid",
    data_format=None,
    dilation_rate=(1, 1),
    groups=1,
    activation=None,
    use_bias=True,
    kernel_initializer="glorot_uniform",
    bias_initializer="zeros",
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

# Conv vs. dense




http://cs231n.github.io/convolutional-networks/

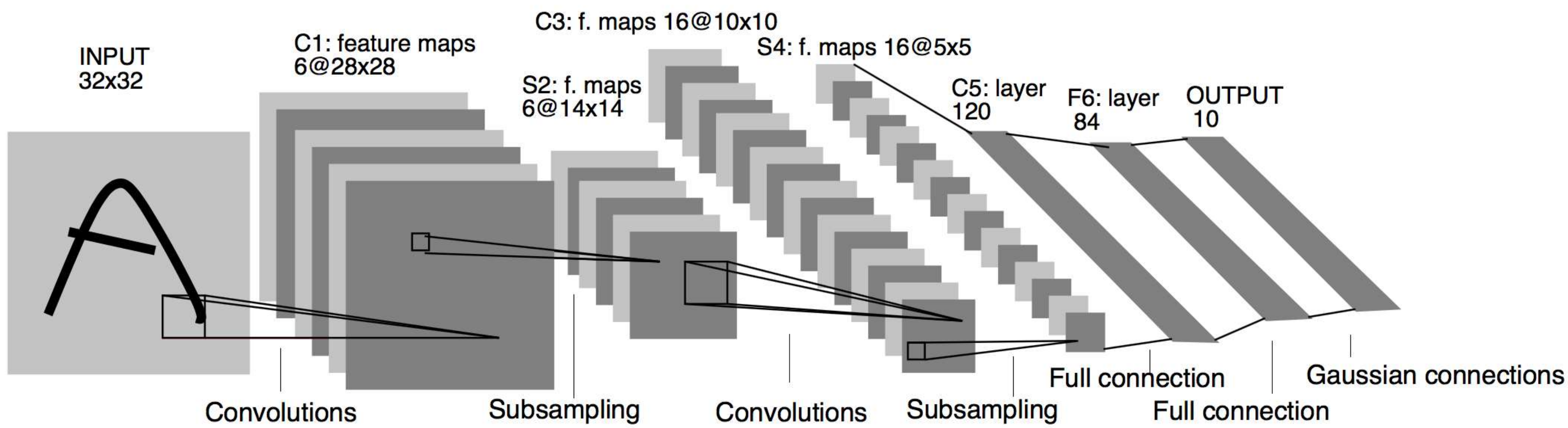
# Max pooling




```
tf.keras.layers.MaxPooling2D(
    pool_size=(2, 2), strides=None, padding="valid", data_format=None, **kwargs
)
```

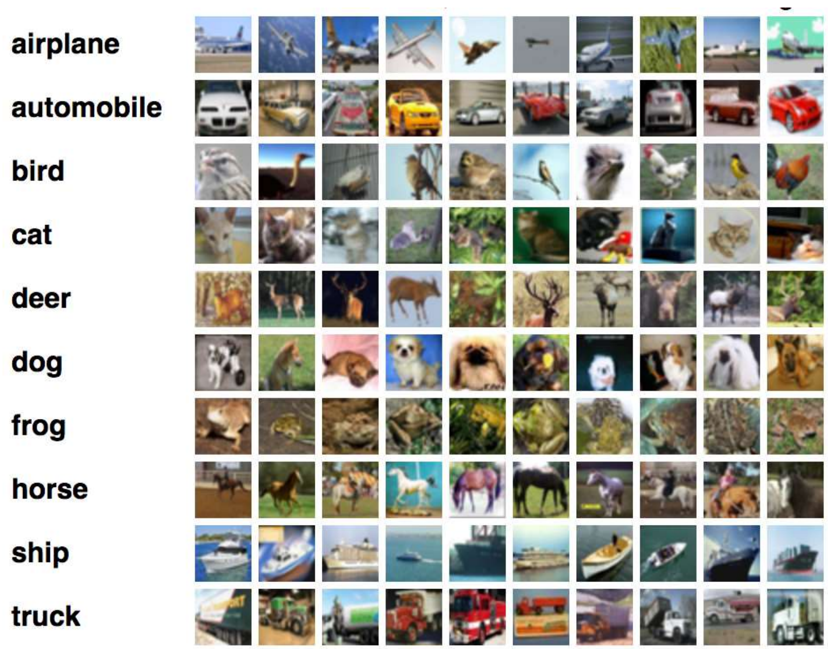
http://cs231n.github.io/convolutional-networks/

# Lenet

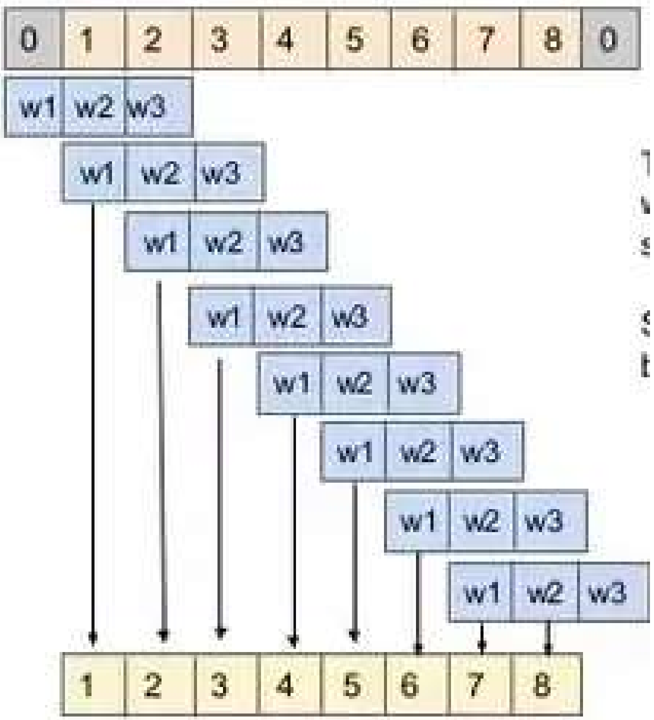



LeCun, Yann, et al. "Gradient-based learning applied to document recognition.
" *Proceedings of the IEEE* 86.11 (1998): 2278-2324.

# Example - cifar10 CNN



# 1D Convolution




```
tf.keras.layers.Conv1D(
    filters,
    kernel_size,
    strides=1,
    padding="valid",
    data_format="channels_last",
    dilation_rate=1,
    groups=1,
    activation=None,
    use_bias=True,
    kernel_initializer="glorot_uniform",
    bias_initializer="zeros",
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

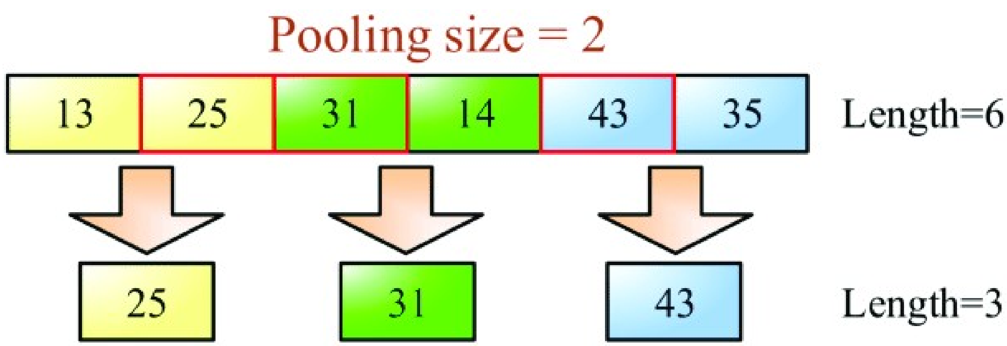
**Input shape**

3+D tensor with shape: `batch_shape + (steps, input_dim)`

**Output shape**

3+D tensor with shape: `batch_shape + (new_steps, filters)` `steps` value might have changed due to padding or strides.

# 1D max pooling




```
tf.keras.layers.MaxPooling1D(
    pool_size=2, strides=None, padding="valid", data_format="channels_last", **kwargs
)
```

**Input shape**

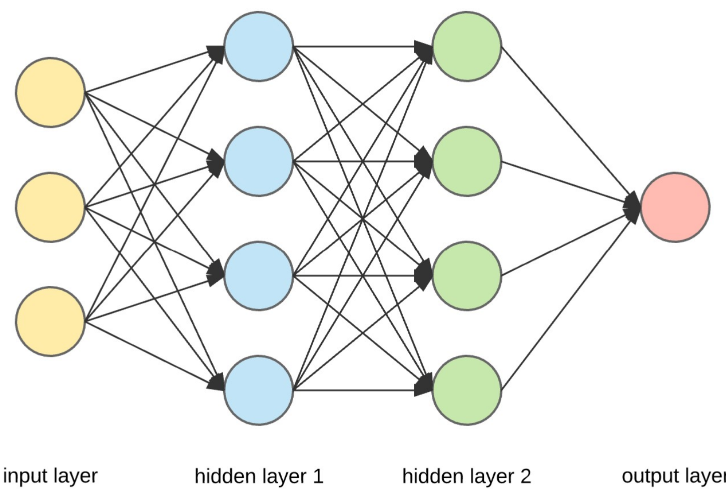
- If `data_format='channels_last'`: 3D tensor with shape `(batch_size, steps, features)`.
- If `data_format='channels_first'`: 3D tensor with shape `(batch_size, features, steps)`.

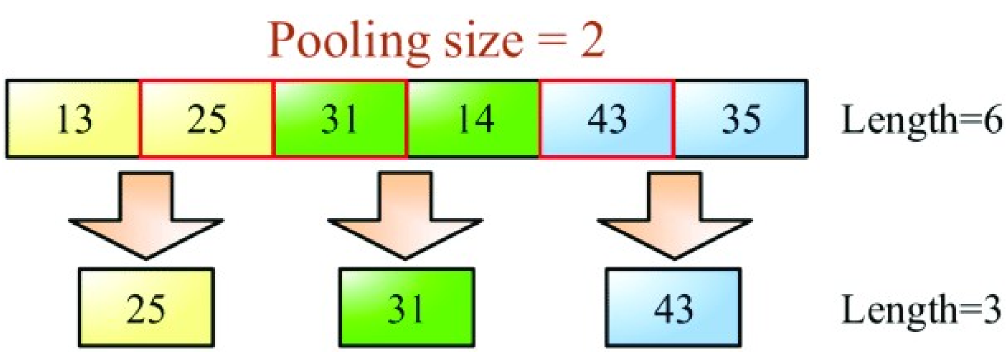
**Output shape**

- If `data_format='channels_last'`: 3D tensor with shape `(batch_size, downsampled_steps, features)`.
- If `data_format='channels_first'`: 3D tensor with shape `(batch_size, features, downsampled_steps)`.

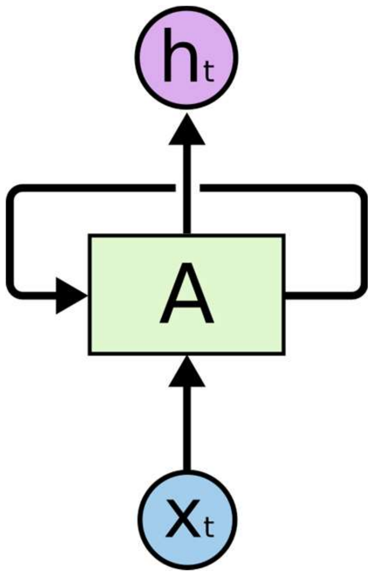
# LSTM - Motivation

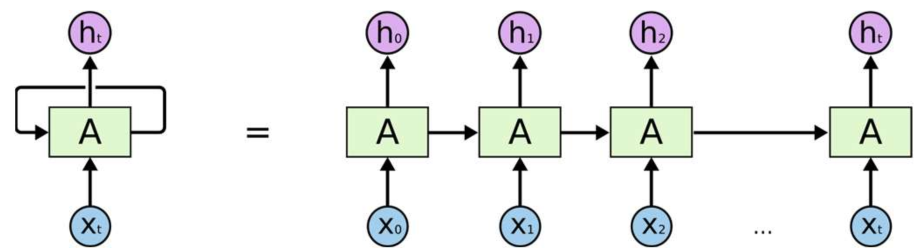
- Text is sequential data
- FNN / Convolution is not sequential
- We need another type of layer



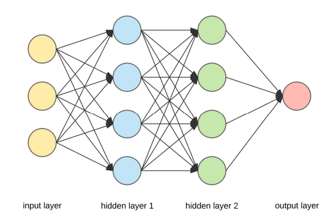



# Recurrent neural network



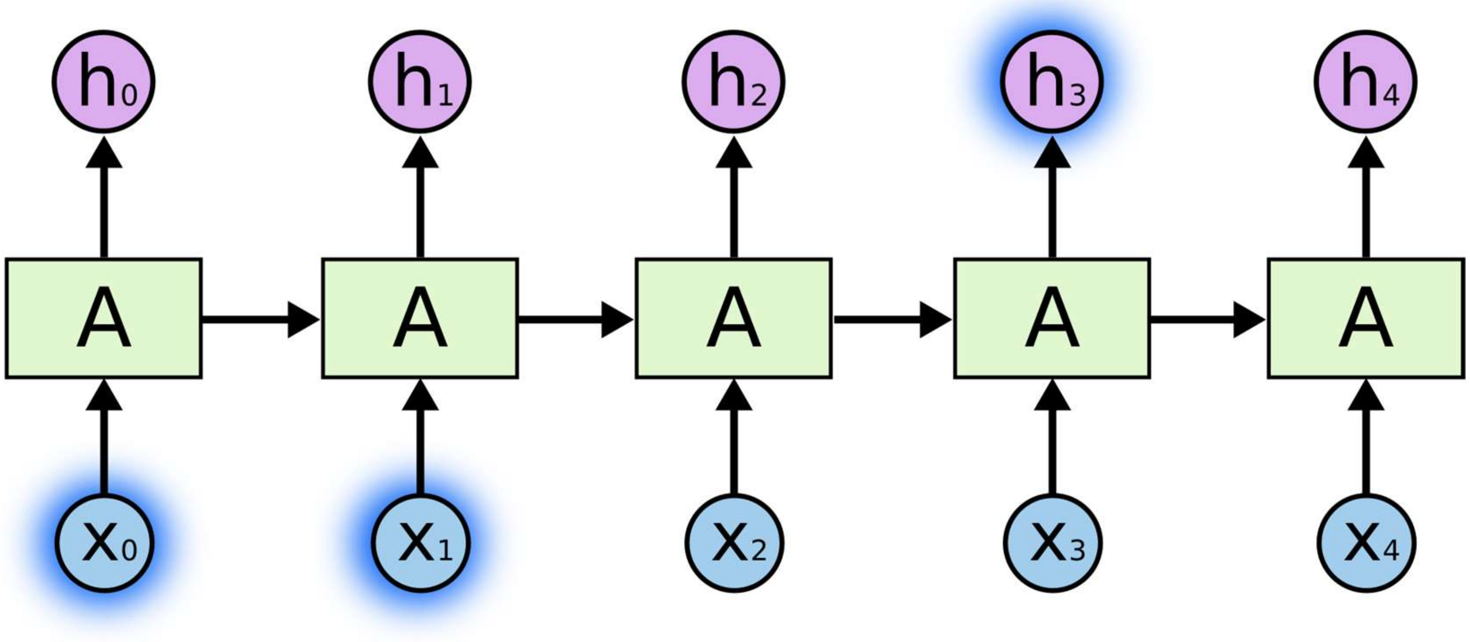




$$\mathbf{h}(t) = \mathbf{x}(t)\mathbf{W}_{in} + \mathbf{h}(t-1)\mathbf{W}$$

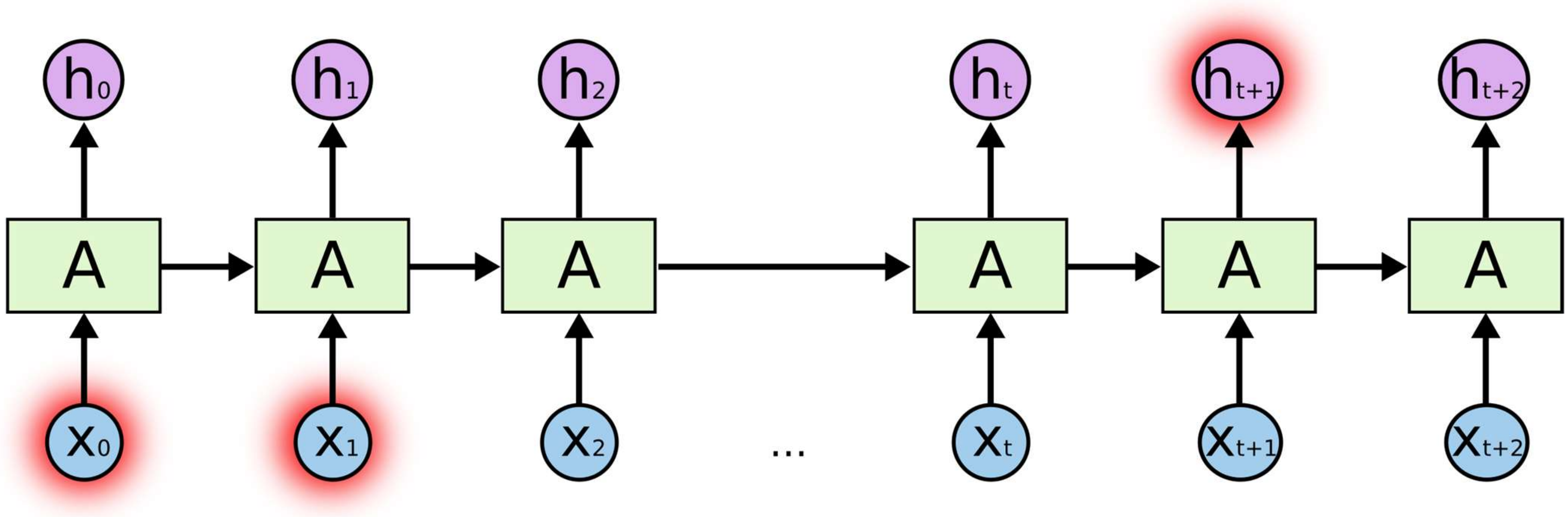



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

# Time dependencies

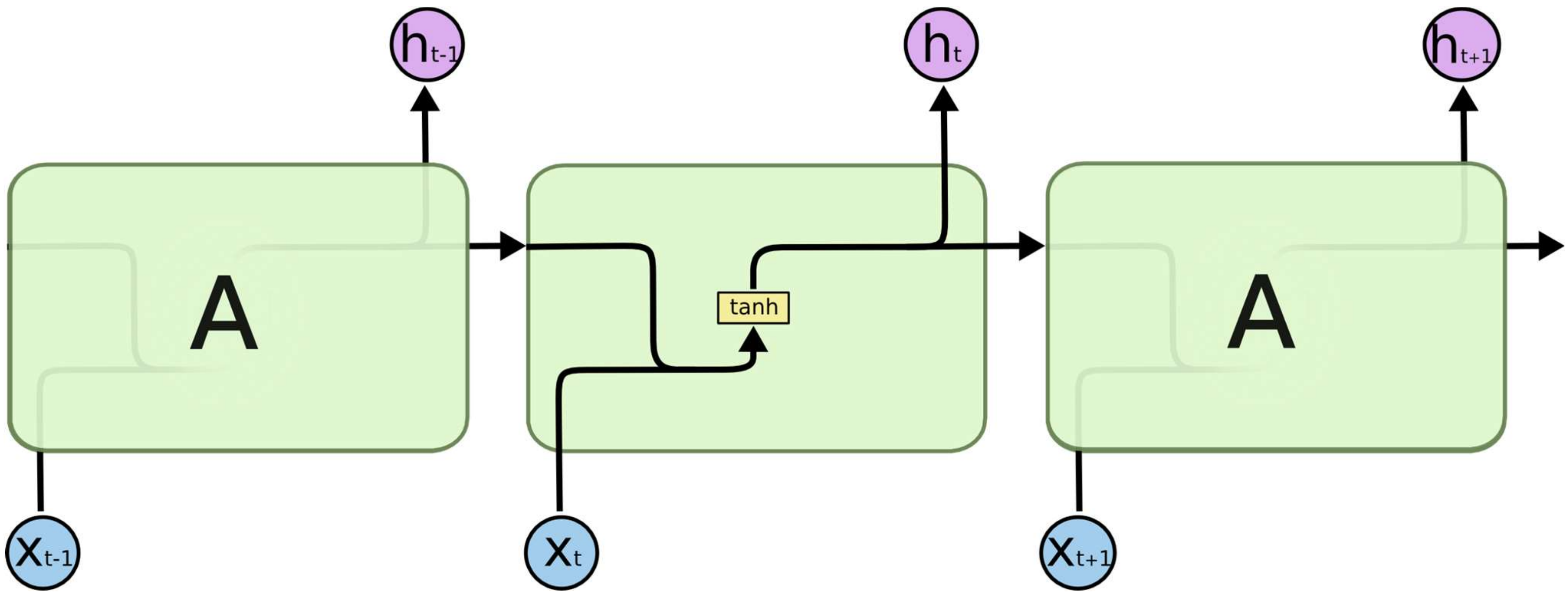


# Long term dependencies

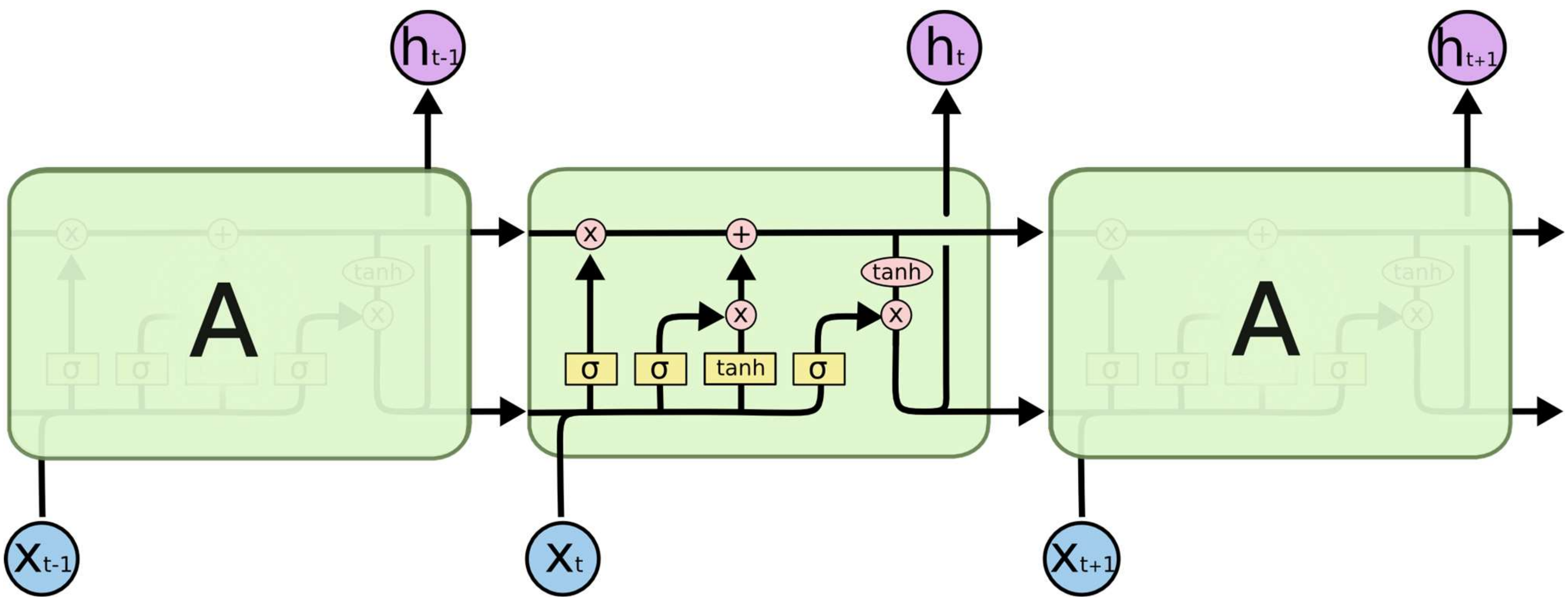



$$\mathbf{h}(t) = \mathbf{x}(t)\mathbf{W}_{in} + \mathbf{h}(t-1)\mathbf{W}$$

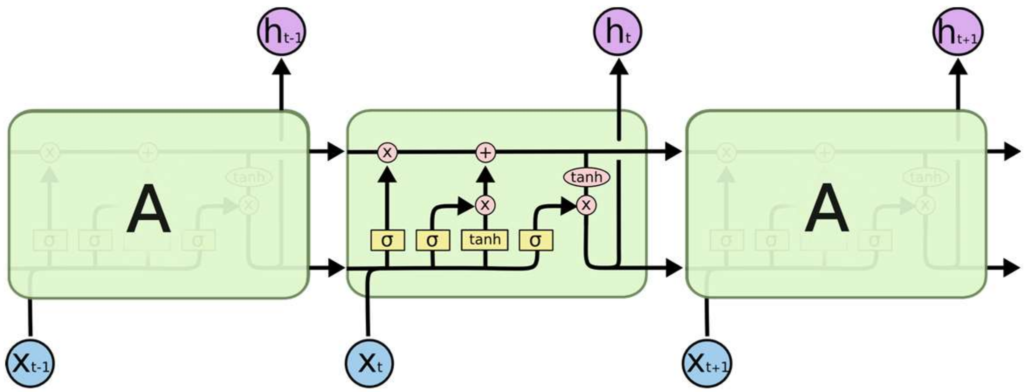
# RNN

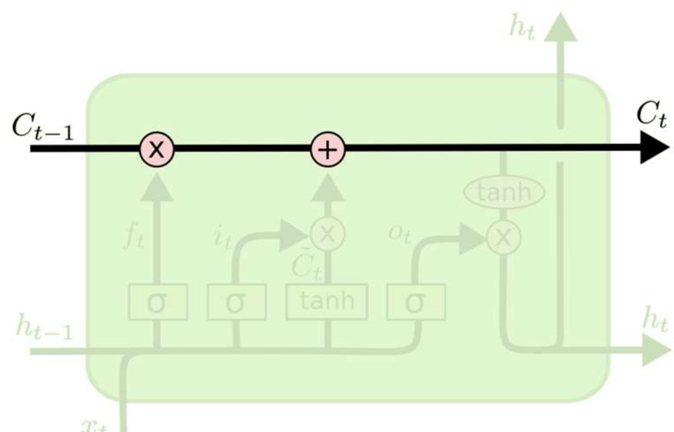


# Long short term memory (LSTM)

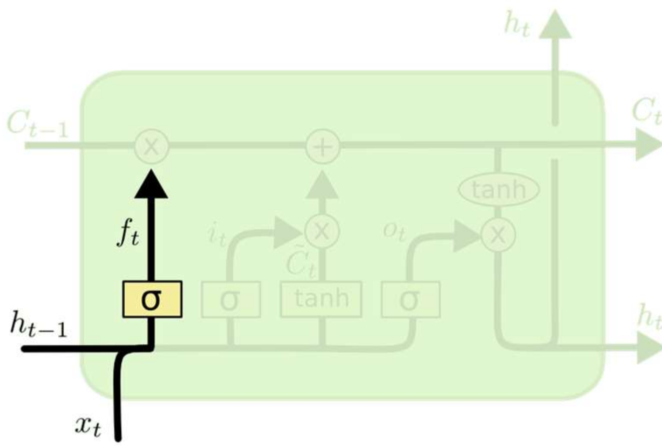


# Cell state



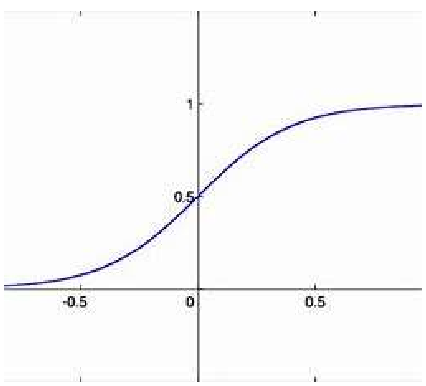



# Forget gate

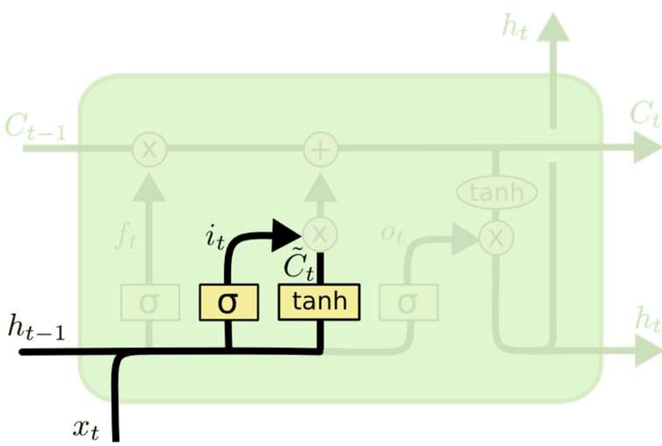



$$\mathbf{h}(t) = \mathbf{x}(t)\mathbf{W}_{in} + \mathbf{h}(t-1)\mathbf{W}$$

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$



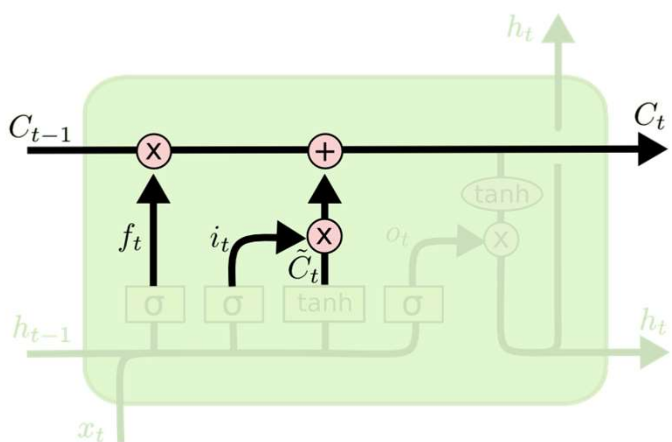
# Input gate




$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

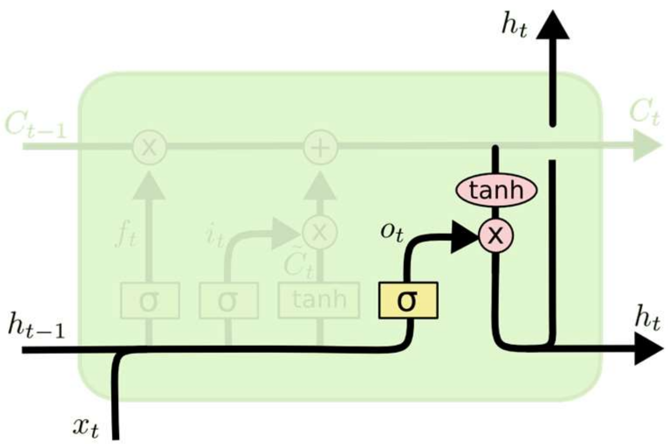
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

# Update cell state




$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

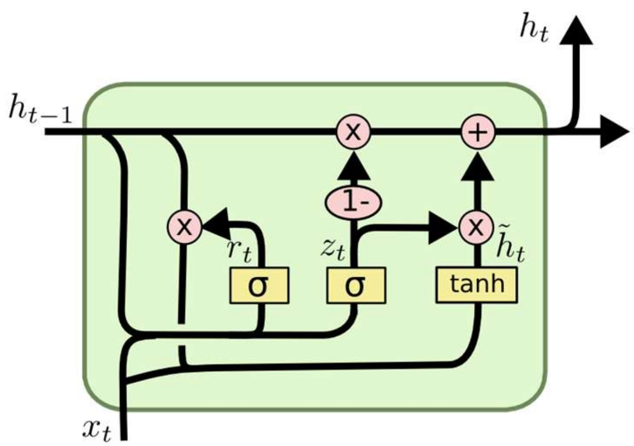
# Output gate




$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

# Gated recurrent unit (GRU)




$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

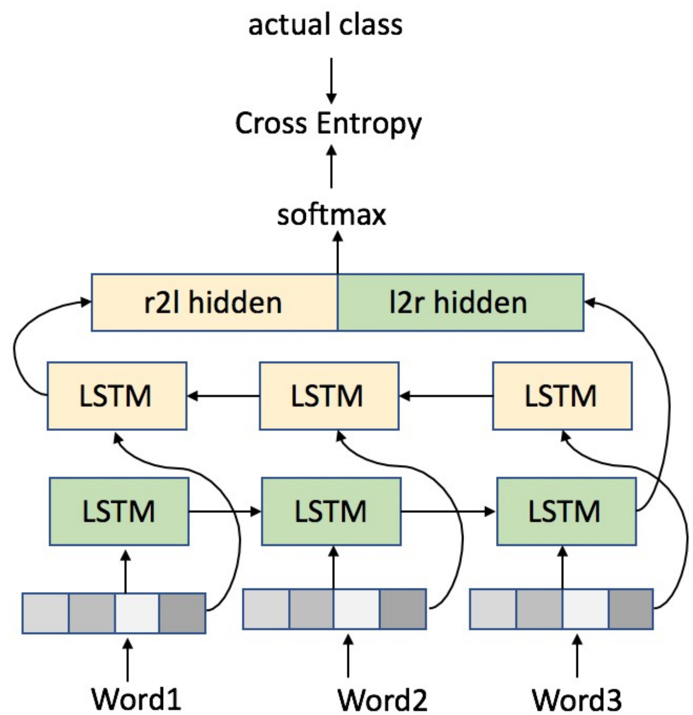
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

# Bidirectional LSTM

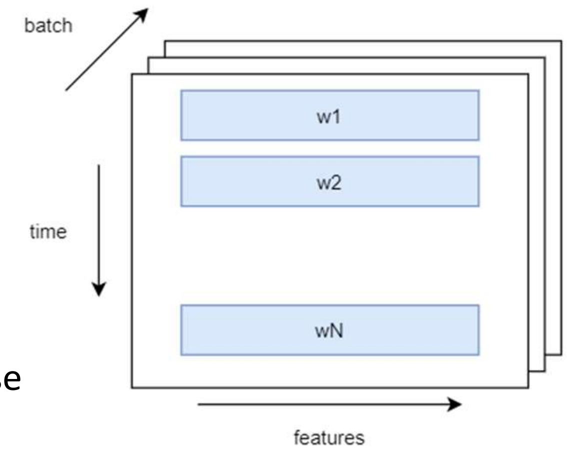
- feed sequence in the forward direction for one flow
- feed sequence backward for the other flow
- concatenate the final state of both flows together



# Shape of tensor for LSTM in Keras

```
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,
    implementation=2,
    return_sequences=False,
    return_state=False,
    go_backwards=False,
    stateful=False,
    time_major=False,
    unroll=False,
    **kwargs
)
```

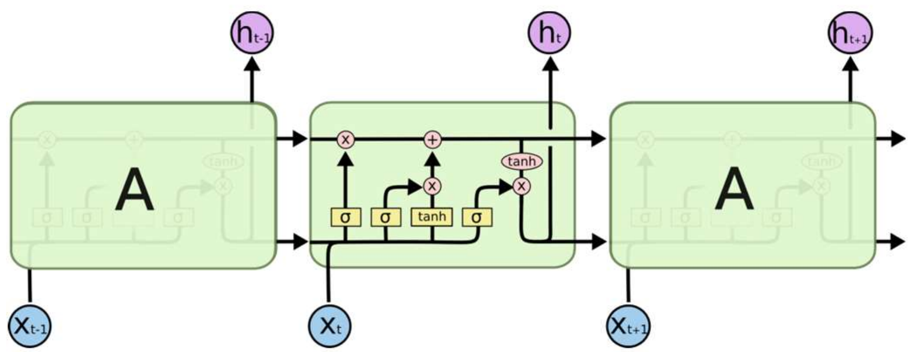
for last layer before dense

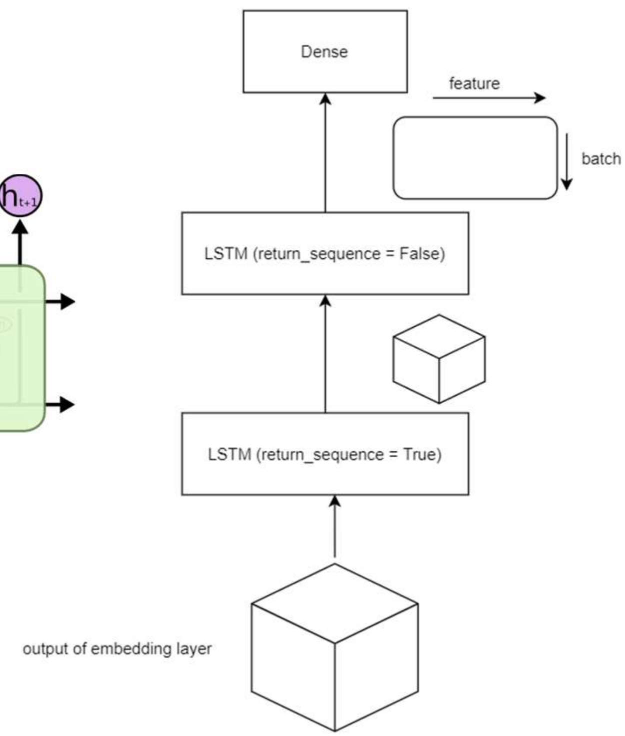



- **inputs**: A 3D tensor with shape `[batch, timesteps, feature]`.
- **mask**: Binary tensor of shape `[batch, timesteps]` indicating whether a given timestep should be masked (optional, defaults to `None`).
- **training**: Python boolean indicating whether the layer should behave in training mode or in inference mode. This argument is passed to the cell when calling it. This is only relevant if `dropout` or `recurrent_dropout` is used (optional, defaults to `None`).
- **initial_state**: List of initial state tensors to be passed to the first call of the cell (optional, defaults to `None` which causes creation of zero-filled initial state tensors).

# Return sequence

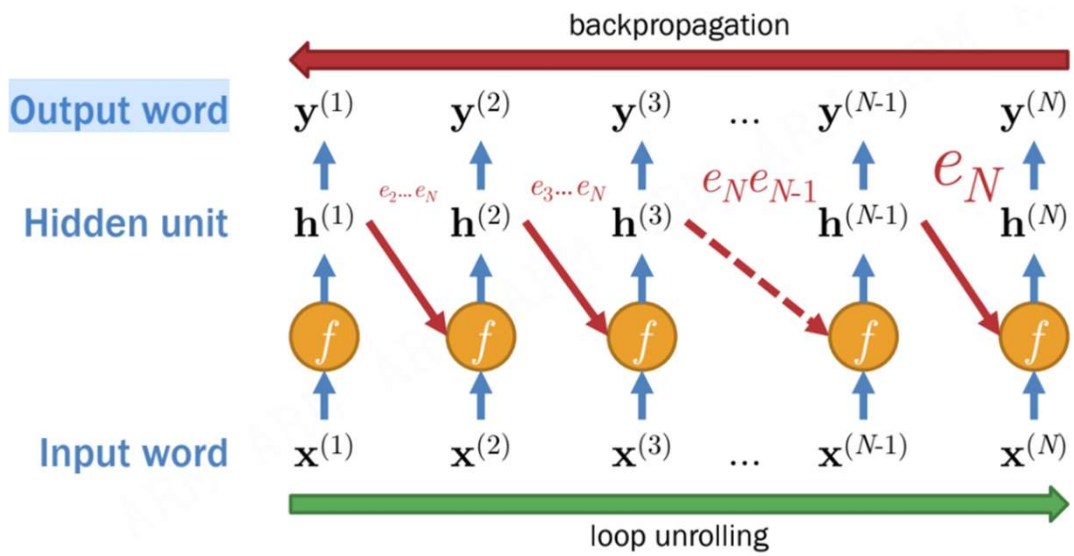
```
tf.keras.layers.LSTM(
    units,
    activation="tanh",
    recurrent_activation="sigmoid",
    use_bias=True,
    kernel_initializer="glorot_uniform",
    recurrent_initializer="orthogonal",
    bias_initializer="zeros",
    unit_forget_bias=True,
    kernel_regularizer=None,
    recurrent_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    recurrent_constraint=None,
    bias_constraint=None,
    dropout=0.0,
    recurrent_dropout=0.0,
    implementation=2,
    return_sequences=False,
    return_state=False,
    go_backwards=False,
    stateful=False,
    time_major=False,
    unroll=False,
    **kwargs
)
```

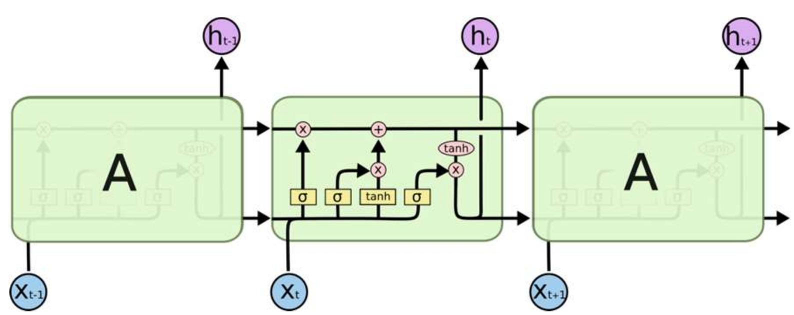


# Training LSTM

# BPTT (backpropagation through time)

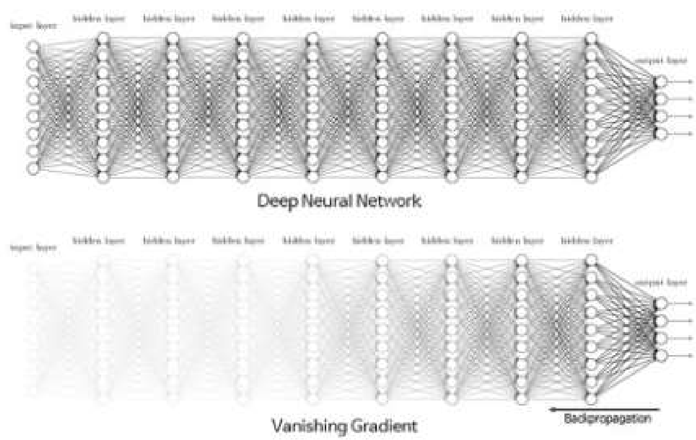


# Vanishing gradient

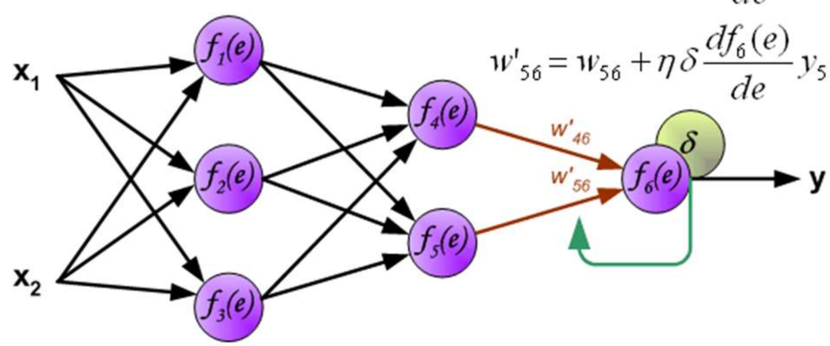
## Sigmoid: Vanishing Gradient Problem


$$\frac{\partial L}{\partial w_{46}} = \frac{\partial L}{\partial \delta} \frac{\partial \delta}{\partial f_6(e_6)} \frac{\partial f_6(e_6)}{\partial e_6} \frac{\partial e_6}{\partial w_{46}}$$




$$w'_{46} = w_{46} + \eta \delta \frac{df_6(e)}{de} y_4$$

$$w'_{56} = w_{56} + \eta \delta \frac{df_6(e)}{de} y_5$$



# Embedding layer

```
tf.keras.layers.Embedding(
    input_dim,
    output_dim,
    embeddings_initializer="uniform",
    embeddings_regularizer=None,
    activity_regularizer=None,
    embeddings_constraint=None,
    mask_zero=False,
    input_length=None,
    **kwargs
)
```

| | t=0 | t=1 | … | t=T |
|---|---|---|---|---|
| text 1 | 13 | 15000 | | 202 |
| text 2 | 3403 | 4550 | | 24384 |
| … | | | | |
| text K | 23433 | 13322 | | 4343 |

**Input shape**

2D tensor with shape: `(batch_size, input_length)`.

**Output shape**

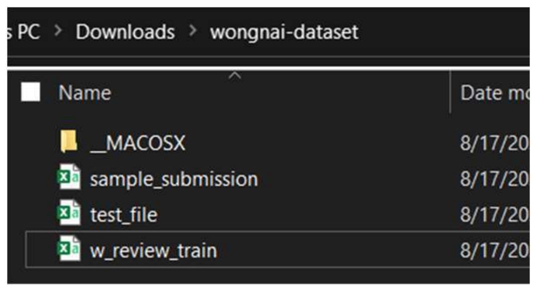
3D tensor with shape: `(batch_size, input_length, output_dim)`.

**Arguments**

- **input_dim**: Integer. Size of the vocabulary, i.e. maximum integer index + 1.
- **output_dim**: Integer. Dimension of the dense embedding.

# Homework 2

- load Wongnai dataset wongnai-data.zip




- Organize the data as Pandas Dataframe

# Homework 2

```
"ร้านอาหารใหญ่มากกกกกกก
เลี้ยวเข้ามาเจอห้องน้ำก่อนเลย เออแปลกดี
ห้องทานหลักๆอยู่ชั้น 2 มีกาแฟ น้ำผึ้ง ซึ่งก็แค่เอาน้ำผึ้งมาราด แพงเวอร์ อย่าสั่งเลย
ลาบไข่ต้ม ไข่มันคาวอะ เลยไม่ประทับใจเท่าไหร่
ทอดมันหัวปลีกรอบอร่อยต้องเบิ้ล
พะแนงห่อไข่อร่อยดี เห้ยแต่ราคา 150บาทมันเกินไปนะ รับไม่ไหวว
เลิกกินแล้วมีขนมหวานให้กินฟรีเล็กน้อย )ขนมไทย)

คงไม่ไปซ้ำ แพงเกิน ";3
```

| label | text |
|---|---|
| 3 | ร้านอาหารใหญ่มากกกกกกก เลี้ยวเข้ามาเจอห้องน้ำก่อนเลย เออแปลกดี ห้องทานหลักๆอยู่ชั้น 2 มีกาแฟ น้ำผึ้ง ซึ่งก็แค่เอาน้ำผึ้งมาราด แพงเวอร์ อย่าสั่งเลย ลาบไข่ต้ม ไข่มันคาวอะ เลยไม่ประทับใจเท่าไหร่ ทอดมันหัวปลีกรอบอร่อยต้องเบิ้ล พะแนงห่อไข่อร่อยดี เห้ยแต่ราคา 150บาทมันเกินไปนะ รับไม่ไหวว เลิกกินแล้วมีขนมหวานให้กินฟรีเล็กน้อย )ขนมไทย) คงไม่ไปซ้ำ แพงเกิน |
| .... | ..... |

<text>;<stars><text>;<stars>... where <stars> is a single character (1-5)

# Homework 2

- Submit as .ipynb file
- The notebook should read *w_review_train.csv* (assume that the .csv file is in the same folder as the .ipynb file)
- cut the text review and the stars value of each review. Then organize into a DataFrame
- In the last cell of the notebook, print the top few rows of the DataFrame using the command .head()
- Hint: google how to read and parse csv file in Python