AuE 8930 Deep Learning: Applications in Engineering

1 Optimization

Loss function

A loss function $\mathcal{L}$ is a function that evaluate the difference between the prediction $\hat{y}$ and the ground truth y. The common loss functions are shown below.

- Least squared error

  $$\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

  which is usually used in the regression problem.

- Cross-entropy

  $$\mathcal{L}(\hat{y}, y) = -\left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right]$$

  which is used for the classification problem.

Cost function

The cost function J is commonly used to assess the performance of a model, and is defined with the loss function $\mathcal{L}$ as follows:

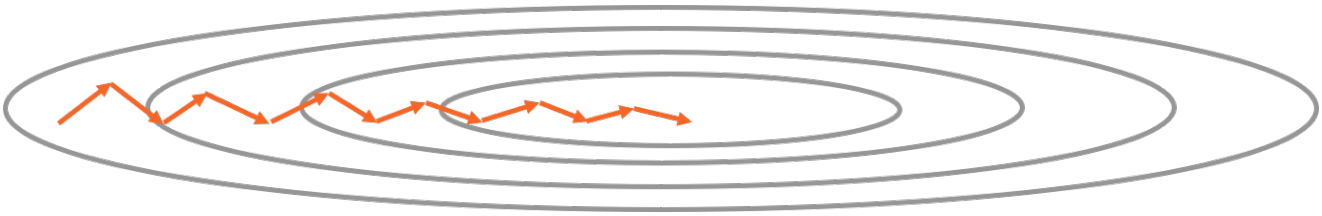
$$J(w, b) = \sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$

Gradient descent

By noting the learning rate $\alpha \in \mathbb{R}$, the update rule for gradient descent can be expressed as follows:

$$w := w - \alpha \cdot \frac{\partial J(w, b)}{\partial w}$$

$$b := b - \alpha \cdot \frac{\partial J(w, b)}{\partial b}$$



Mini-batch gradient descent (MBGD)

During the training phase, updating weights is usually not based on the whole training set at once due to computation complexities or one data point due to noise issues. Instead, the update step is done on mini-batches, where the batch size is a hyperparameter that we can tune.

- If batch size is set to be 1, we will get **stochastic gradient descent (SGD)**.
- If batch size equals the number of training examples, we are using **batch gradient descent (BGD)**.

Adaptive learning rate

Letting the learning rate vary when training a model can reduce the training time and improve the numerical optimal solution.

- Momentum
- RMSprop
- Adam (most commonly used technique)

Learning rate decay

Learning rate decay is another method to speed up the training process.

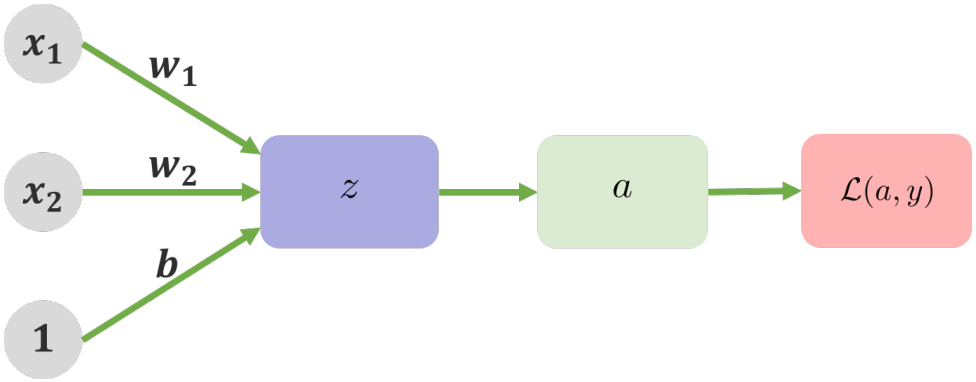
- Exponential decay
- Discrete staircase

---

2 Logistic Regression

Logistic regression models the probabilities for classification problems with two possible outcomes. It's an extension of the linear regression model for classification problems.

Forward propagation

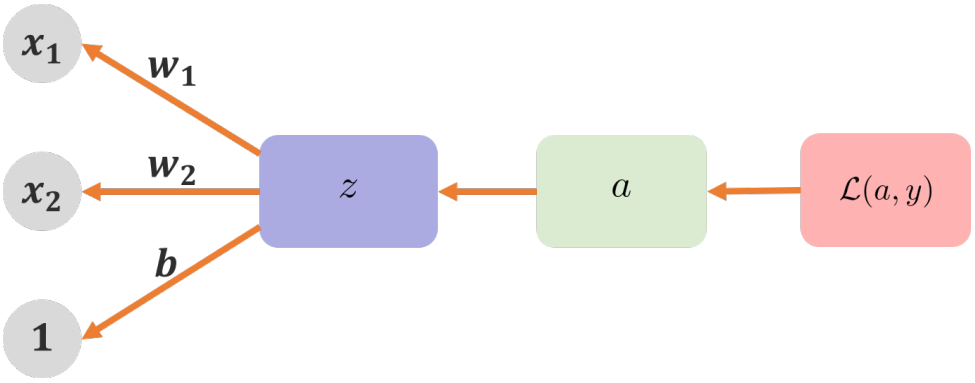


$$z = w_1 x_1 + w_2 x_2 + b = w^T x + b$$

$$\hat{y} = a = \sigma(z) = \frac{1}{1 + e^{-(w^T x + b)}}$$

$$\mathcal{L}(\hat{y}, y) = -\left(y \log(\hat{y}) + (1-y)\log(1-\hat{y})\right) = -\left(y \log \frac{1}{1+e^{-(w^T x+b)}} + (1-y)\log\left(1 - \frac{1}{1+e^{-(w^T x+b)}}\right)\right)$$

Backward propagation



$$\mathrm{d}a = \frac{\mathrm{d}L}{\mathrm{d}a} = \frac{-y}{a} + \frac{1-y}{1-a}$$

$$\mathrm{d}z = \frac{\mathrm{d}L}{\mathrm{d}z} = \frac{\mathrm{d}L}{\mathrm{d}a} \cdot \frac{\mathrm{d}a}{\mathrm{d}z} = a - y$$

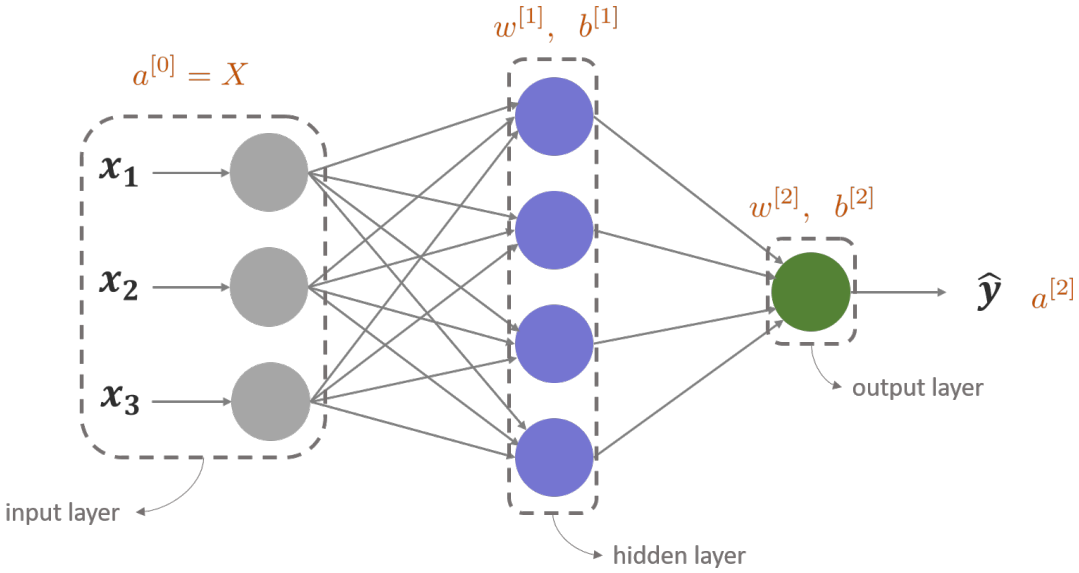
$$\mathrm{d}w_1 = \frac{\mathrm{d}L}{\mathrm{d}w_1} = \frac{\mathrm{d}L}{\mathrm{d}z} \cdot \frac{\mathrm{d}z}{\mathrm{d}w_1} = (a-y)x_1$$

$$\mathrm{d}w_2 = \frac{\mathrm{d}L}{\mathrm{d}w_2} = \frac{\mathrm{d}L}{\mathrm{d}z} \cdot \frac{\mathrm{d}z}{\mathrm{d}w_2} = (a-y)x_2$$

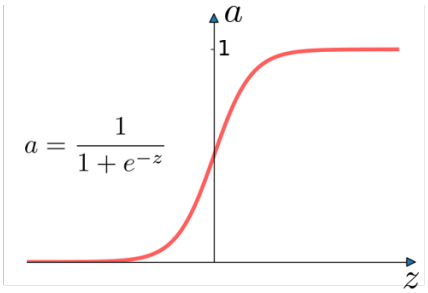
$$\mathrm{d}b = \frac{\mathrm{d}L}{\mathrm{d}b} = \frac{\mathrm{d}L}{\mathrm{d}z} \cdot \frac{\mathrm{d}z}{\mathrm{d}b} = a - y$$

---

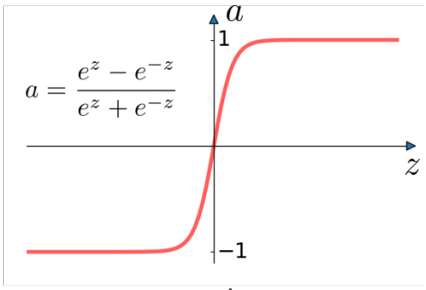
3 Shallow neural network



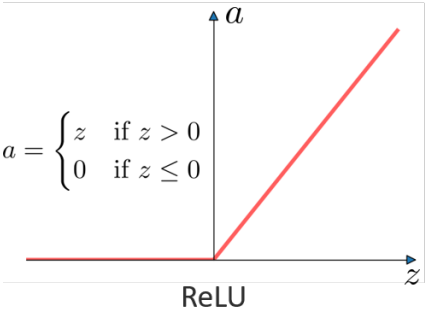
Common activation functions



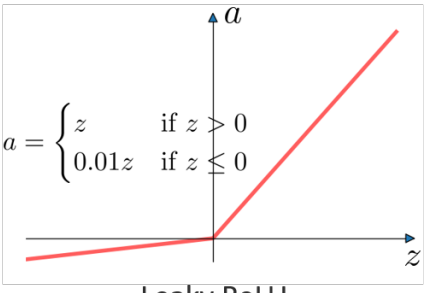
Sigmoid



tanh



ReLU



Leaky ReLU

Weights initialization

Instead of initializing the weights in a purely random manner, we should have initial weights that take into account characteristics that are unique to the architecture.

- He method: $w^{[l]}$ = np.random.randn(shape) * np.sqrt($\frac{2}{n^{(l-1)}}$)
- Xavier method: $w^{[l]}$ = np.random.randn(shape) * np.sqrt($\frac{1}{n^{(l-1)}}$)
- Other method: $w^{[l]}$ = np.random.randn(shape) * np.sqrt($\frac{2}{n^{(l-1)}+n^{(l)}}$)

Forward propagation

$$Z^{[1]} = W^{[1]} X + b^{[1]}$$

$$A^{[1]} = g^{[1]}\left(Z^{[1]}\right)$$

$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$

$$A^{[2]} = g^{[2]}\left(Z^{[2]}\right)$$

Backward propagation

$$\mathrm{d}Z^{[2]} = A^{[2]} - Y$$

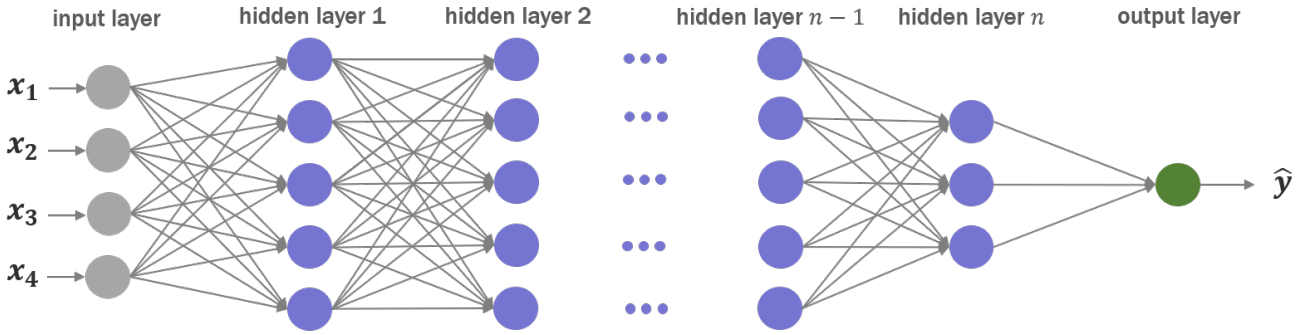
$$\mathrm{d}W^{[2]} = \frac{1}{m}\mathrm{d}Z^{[2]} {A^{[1]}}^T$$

$$\mathrm{d}b^{[2]} = \frac{1}{m}\,\mathrm{np.sum}(\mathrm{d}Z^{[2]}, \mathrm{axis} = 1, \mathrm{keepdims=True})$$

$$\mathrm{d}Z^{[1]} = {W^{[2]}}^T \mathrm{d}Z^{[2]} * {g^{[1]}}'\left(Z^{[1]}\right)$$

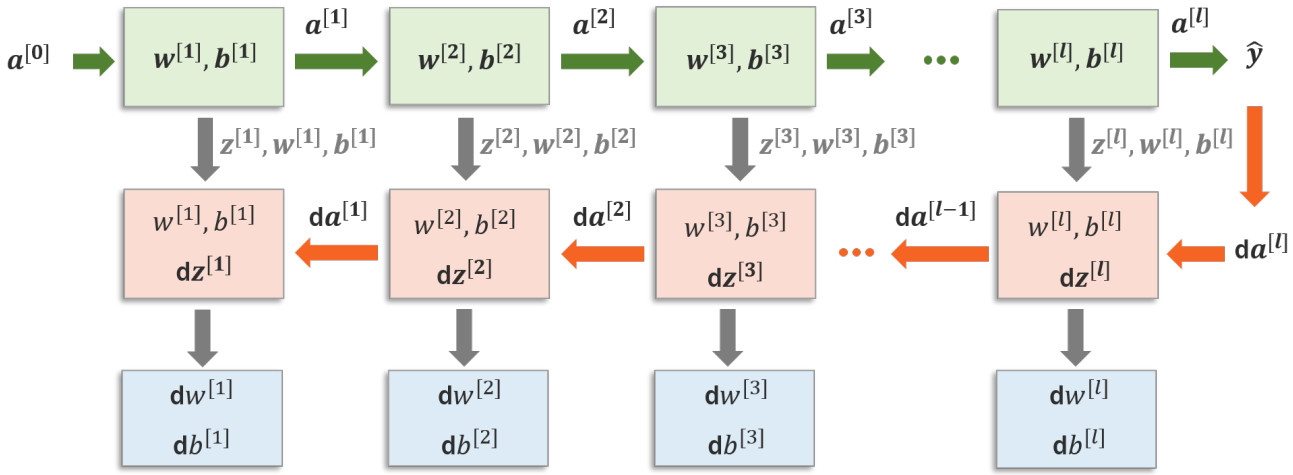
---

4 Deep neural network



$n+1$ layer NN

Building blocks



Forward propagation

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[1]}$$

$$A^{[l]} = g^{[l]}\left(Z^{[l]}\right)$$

Backward propagation

$$\mathrm{d}Z^{[l]} = \mathrm{d}A^{[l]} * {g^{[l]}}'\left(Z^{[l]}\right)$$

$$\mathrm{d}W^{[l]} = \frac{1}{m}\mathrm{d}Z^{[l]} \cdot A^{[l-1]T}$$

$$\mathrm{d}b^{[l]} = \frac{1}{m}\,\mathrm{np.sum}(\mathrm{d}Z^{[l]}, \mathrm{axis} = 1, \mathrm{keepdims}{=}\mathrm{True})$$

$$\mathrm{d}A^{[l-1]} = W^{[l]T}\mathrm{d}Z^{[l]}$$

Updating parameters

$$w^{[l]} := w^{[l]} - \alpha \cdot \mathrm{d}w^{[l]}$$

$$b^{[l]} := b^{[l]} - \alpha \cdot \mathrm{d}b^{[l]}$$

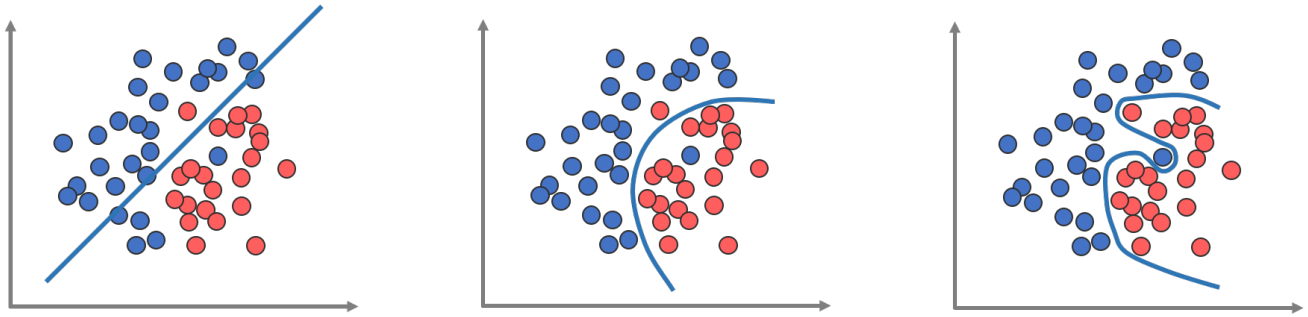
Hyperparameters

1. learning rate α
2. # of iterations of gradient descent process
3. # of hidden layers L
4. # of hidden units in each layer
5. choice of activation function

---

5 Tips and tricks in NN

Bias and variance



Some cases of bias and variance:

Error	High bias (underfitting)	High variance (overfitting)	High bias and high variance
training set	14%	2%	15%
dev set	15%	12%	25%

Methods to reduce high bias:

- increase model complexity
    - more # of hidden layers
    - more neurons in each layer
- increase the number of features
- training longer (more epochs)
- use better optimization techniques
- remove noise from data

Approaches to reduce high variance:

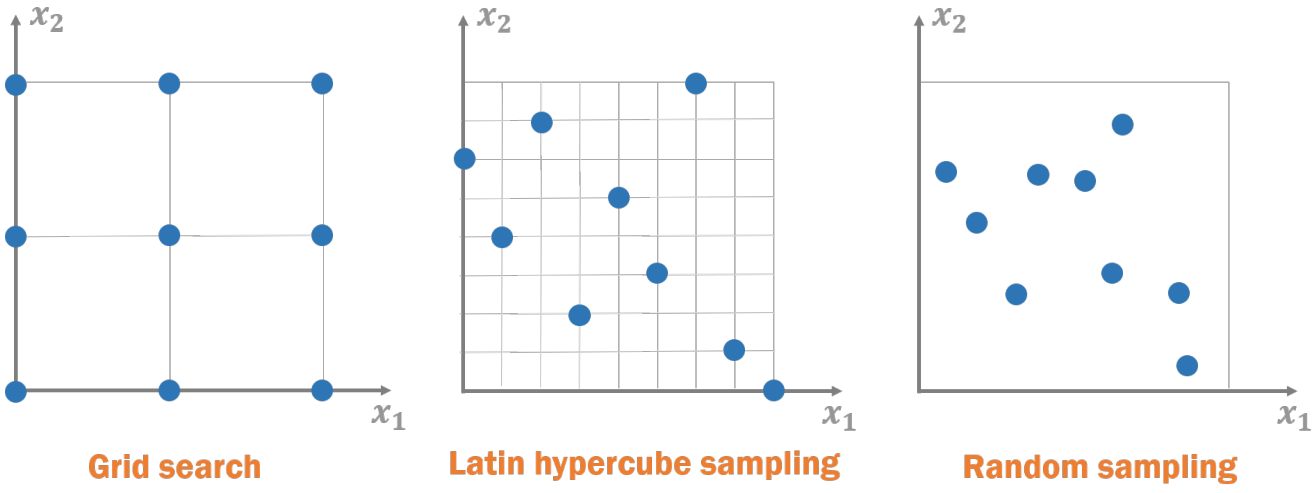
- increase the training data set
- reduce model complexity
- regularization techniques
- early stopping during training phase

Regularization

- Dropout
- Weight regularization
    - L1
    - L2
- Early stopping
- Input normalization

Hyperparameter tuning

Strategies to sample the hyperparameter design space:



Batch normalization

Batch normalization is a method used to make NNs faster and more stable through normalization of the input layer by re-centering and re-scaling. The algorithm is shown below.

Given a batch of data in NN

$$\text{compute} \quad z^{(1)}, z^{(2)}, \ldots, z^{(m)}$$

$$\mu = \frac{1}{m}\sum_{i=1}^{m} z^{(i)}$$

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}\left(z^{(i)} - \mu\right)^2$$

$$z^{(i)}_{\text{normlization}} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$\tilde{z}^{(i)}_{\text{normlization}} = \gamma z^{(i)}_{\text{normalization}} + \beta$$

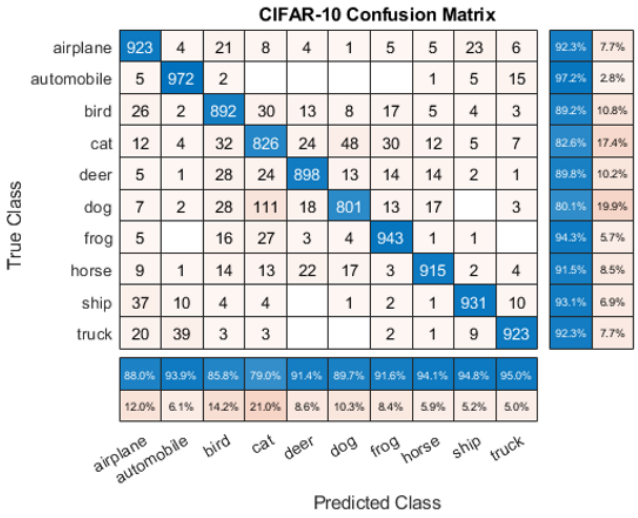
γ and β are learnable parameters in the algorithm

Avoid the mean value of $z^{(i)}_{\text{normlization}}$ close to 0

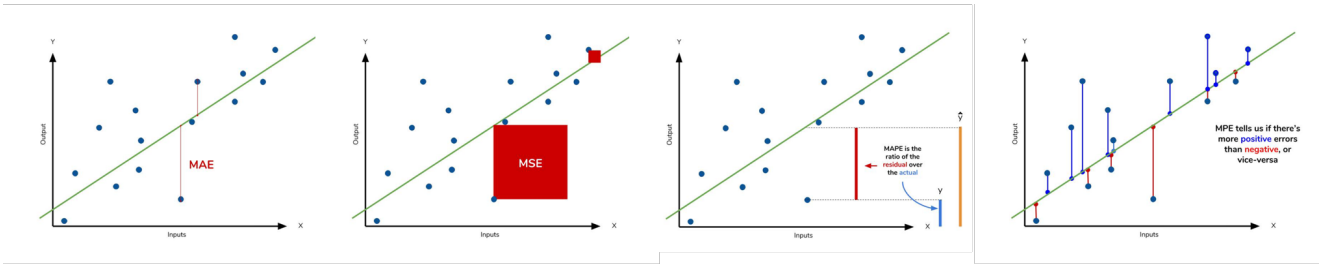
Classifier performance

Confusion matrix is the most commonly used error measurement in classification problem.

- A row-normalized row summary displays the number of correctly and incorrectly classified observations for each true class as percentages of the number of observations of the corresponding true class.
- A column-normalized column summary displays the number of correctly and incorrectly classified observations for each predicted class as percentages of the number of observations of the corresponding predicted class.



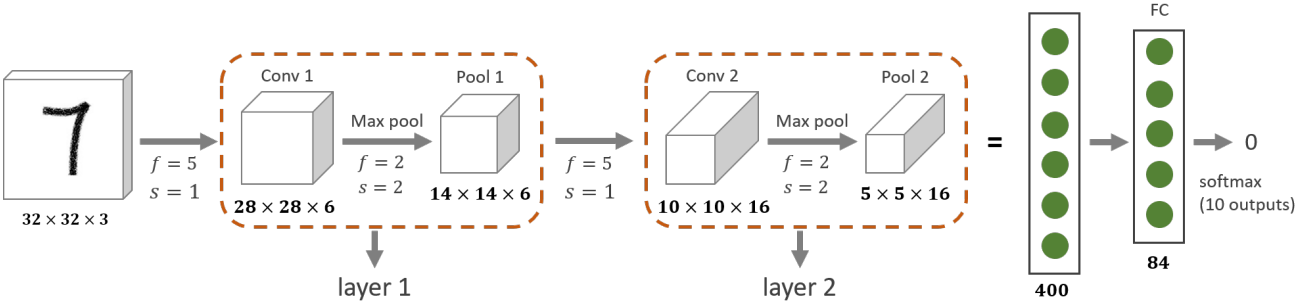
Regressor performance



6 Convolutional neural network

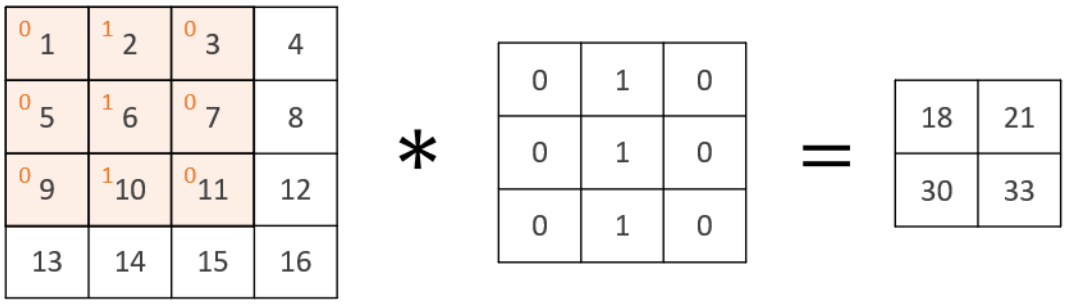
Architecture of CNN

Convolutional neural networks, also known as CNNs, are a specific type of neural networks that are generally composed of the following layers:

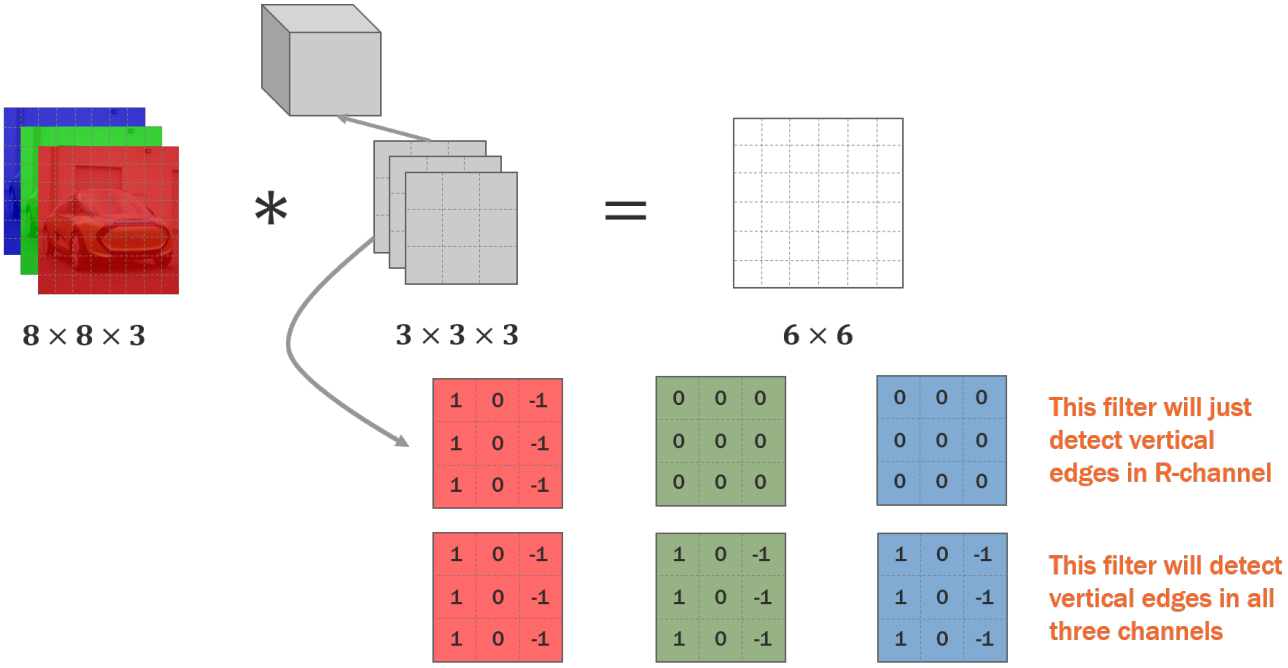


Convolution layer

The convolution layer (CONV) uses filters that perform convolution operations as it is scanning the input I with respect to its dimensions. The resulting output is called *feature map*.



Convolution on volume



Padding

The issue when applying convolutional layers is that we tend to lose pixels on the edges of the image. A straightforward solution to this problem is to add extra pixels of filler around the boundary of our input image.

0 0	1 0	0 0	0	0	0
0 0	1 1	0 2	3	4	0
0 0	1 5	0 6	7	8	0
0	9	10	11	12	0
0	13	14	15	16	0
0	0	0	0	0	0

$*$

0	1	0
0	1	0
0	1	0

$=$

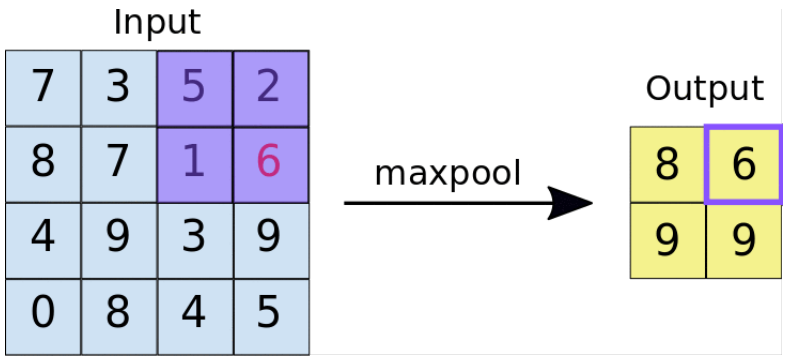
6	8	10	12
15	18	21	24
27	30	33	36
22	24	26	28

Computing output shape

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

Max pooling

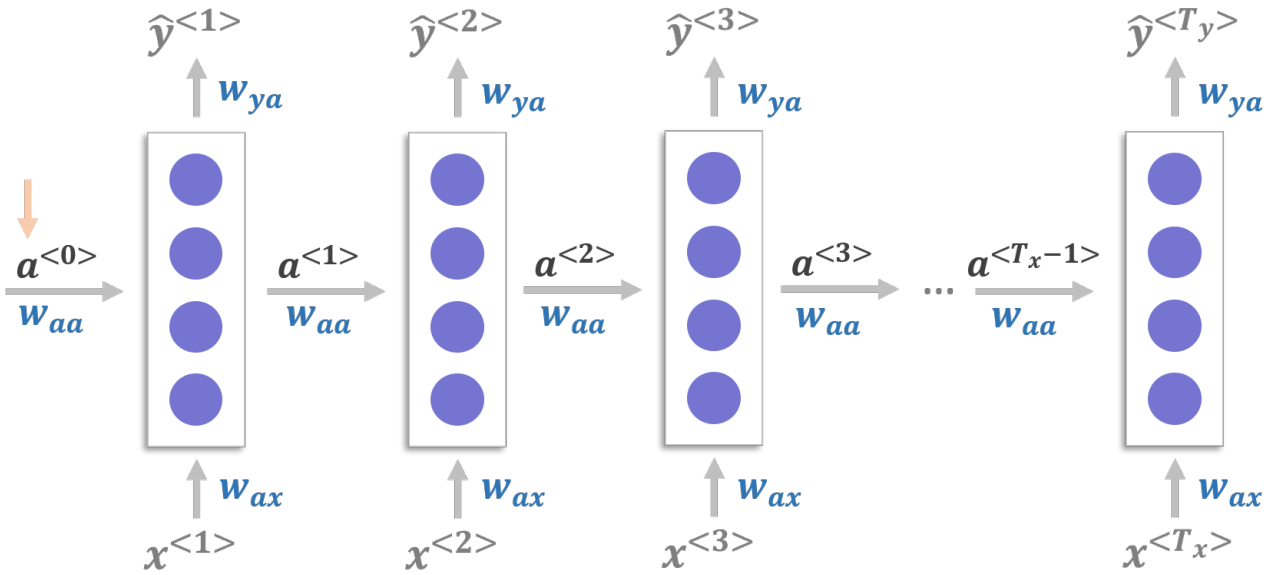
The max pooling layer is a downsampling operation, typically applied after a convolution layer, which does some spatial invariance.



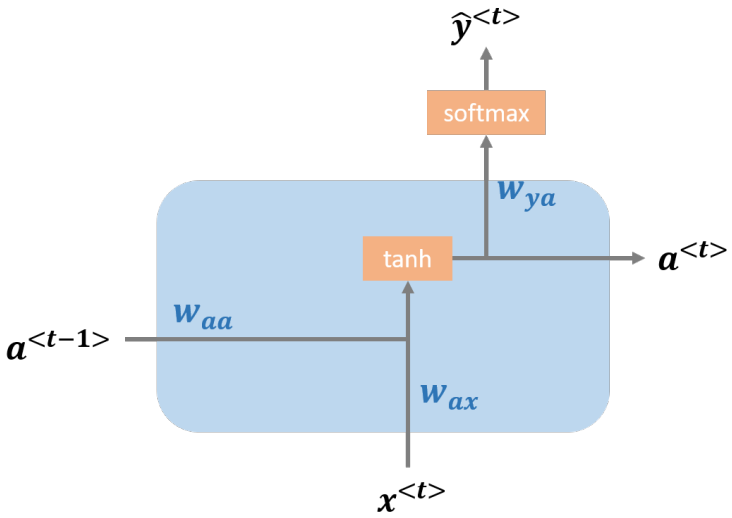
7 Recurrent neural network

Architecture of RNN

Recurrent neural networks, also known as RNNs, are a class of neural networks that allow previous outputs to be used as inputs while having hidden states.



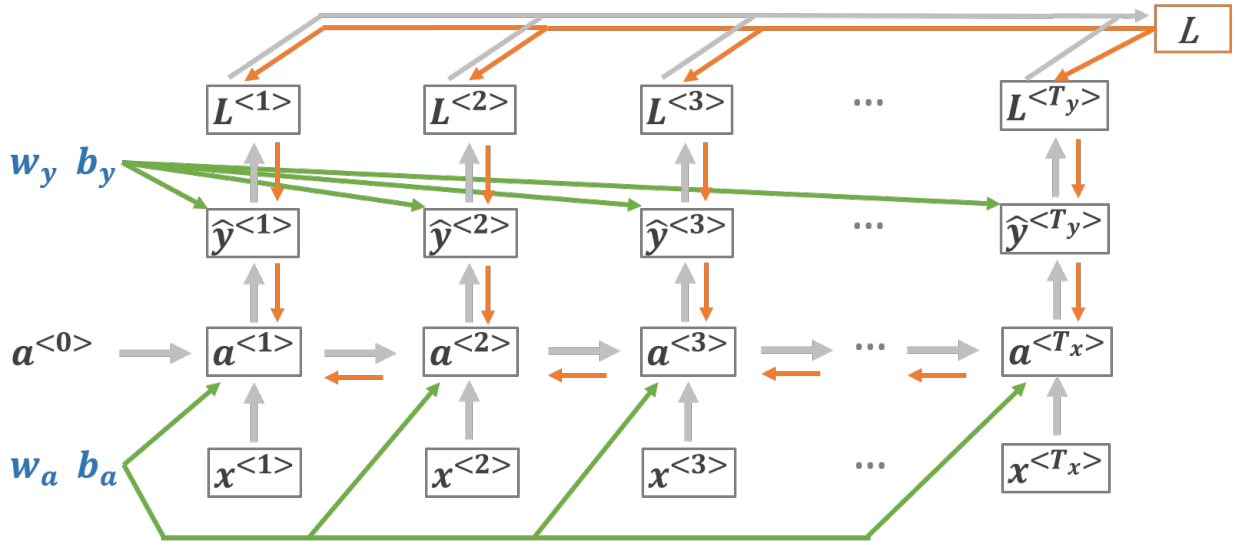
Forward propagation



$$a^{<t>} = g\left(w_{aa} \cdot a^{<t-1>} + w_{ax} \cdot x^{<t>} + b_a\right)$$

$$\hat{y}^{<t>} = g\left(w_{ya} \cdot a^{<t>} + b_y\right)$$

Backward propagation

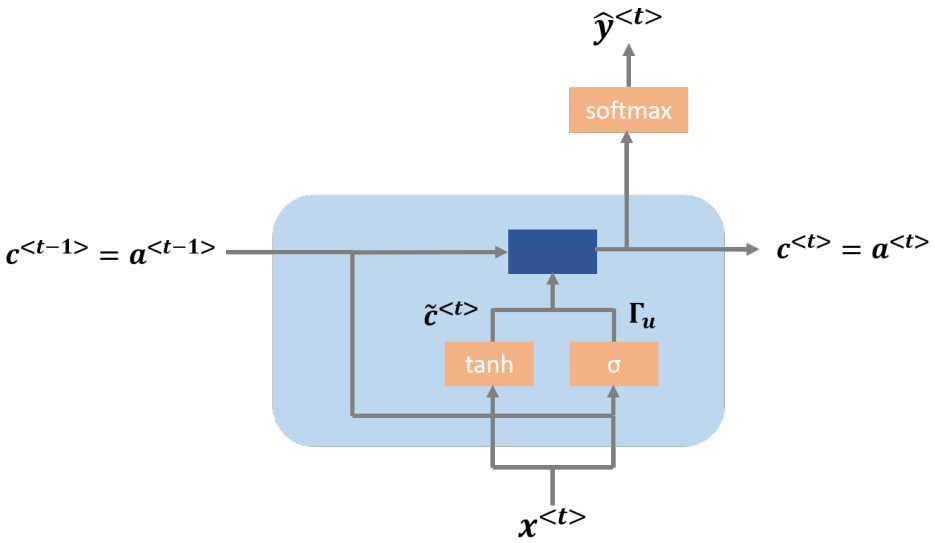


$$\mathcal{L}^{<t>}\left(\hat{y}^{<t>}, y^{<t>}\right) = -y^{<t>}\log\left(\hat{y}^{<t>}\right) - \left(1 - y^{<t>}\right)\log\left(1 - \hat{y}^{<t>}\right)$$

$$\mathcal{L}\left(\hat{y}, y\right) = \sum_{t=1}^{T_x} \mathcal{L}^{<t>}\left(\hat{y}^{<t>}, y^{<t>}\right)$$

Gated recurrent unit (GRU)

The key distinction between vanilla RNNs and GRUs is that the latter support gating of the hidden state. This means that we have dedicated mechanisms for when a hidden state should be *updated* and also when it should be *reset*.



$$\tilde{c}^{<t>} = \tanh\left(w_c\left[c^{<t-1>}, x^{<t>}\right] + b_c\right)$$

$$\Gamma_u = \sigma\left(W_u\left[c^{<t-1>}, x^{<t>}\right] + b_u\right)$$

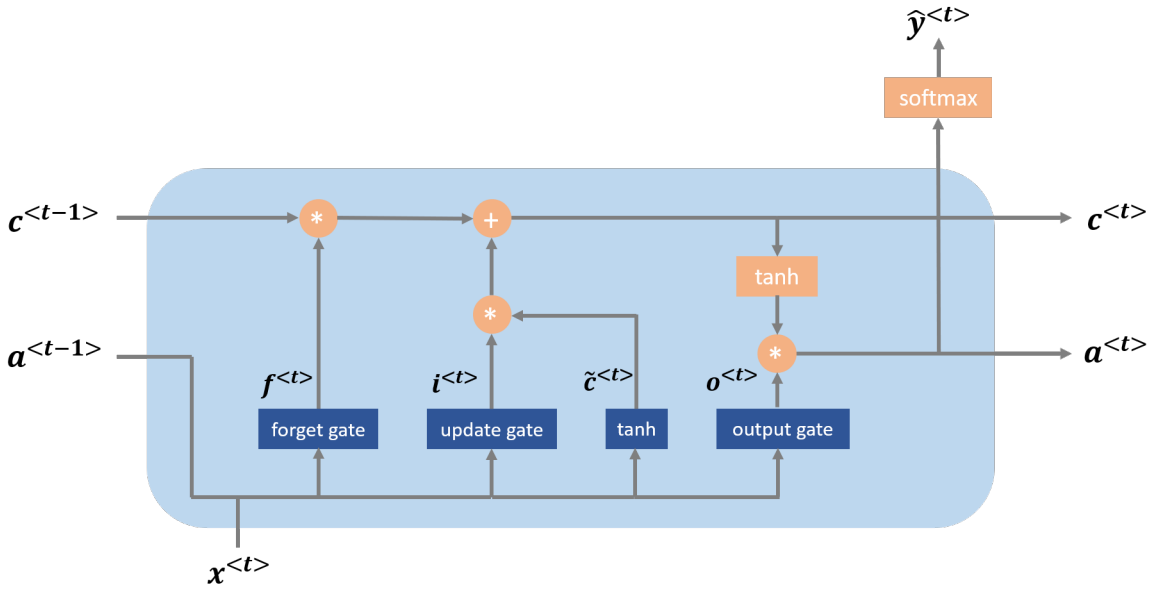
$$\Gamma_r = \sigma\left(W_r\left[c^{<t-1>}, x^{<t>}\right] + b_r\right)$$

$$a^{<t>} = c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

Long short-term memory (LSTM)

LSTM introduces a *memory cell* (or *cell* for short) that has the same shape as the hidden state (some literatures consider the memory cell as a special type of the hidden state), engineered to record additional information.

- **output gate**: read out the entries from the cell.
- **update gate** (input gate): decide when to read data into the cell.
- **forget gate**: reset the content of the cell.



$$\tilde{c}^{<t>} = \tanh\left(W_c\left[a^{<t-1>}, x^{<t>}\right] + b_c\right)$$

$$\Gamma_u = \sigma\left(W_u\left[a^{<t-1>}, x^{<t>}\right] + b_u\right)$$

$$\Gamma_f = \sigma\left(W_f\left[a^{<t-1>}, x^{<t>}\right] + b_f\right)$$

$$\Gamma_o = \sigma\left(W_o\left[a^{<t-1>}, x^{<t>}\right] + b_o\right)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

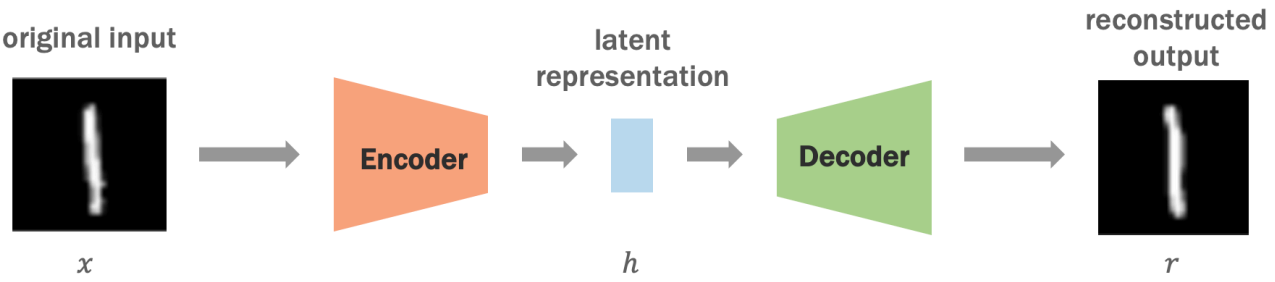
$$a^{<t>} = \Gamma_o * c^{<t>}$$

8 Autoencoders and Variational Autoencoders

Unsupervised learning

- clustering
- data compression
- representation learning

Autoencoders

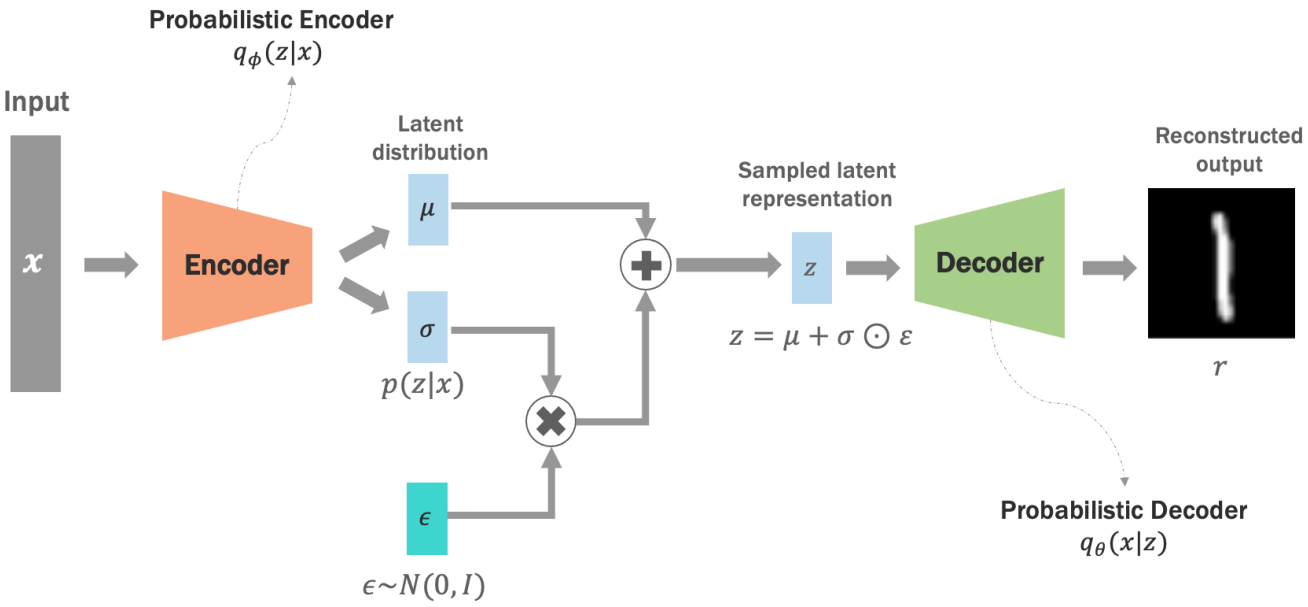


- Encoder: Compact/compress input features into a latent-space features of usually smaller dimension.
- Decoder: Reconstruct original input from the latent space.

Common applications of autoencoder:

- denoising
- image colorization
- image compression
- image search and retrieval

Variational Autoencoders (Optional)



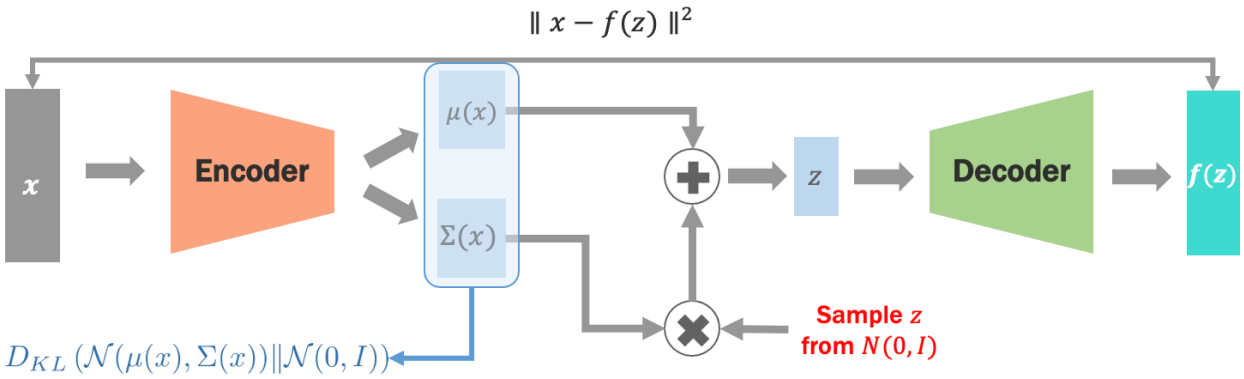
- KL divergence

$$D_{KL}\left(Q(z|x)||P(z|x)\right) = -\sum_{z} Q(z|x)\log\frac{P(x,z)}{Q(z|x)} + \log P(x)$$

- ELBO loss function

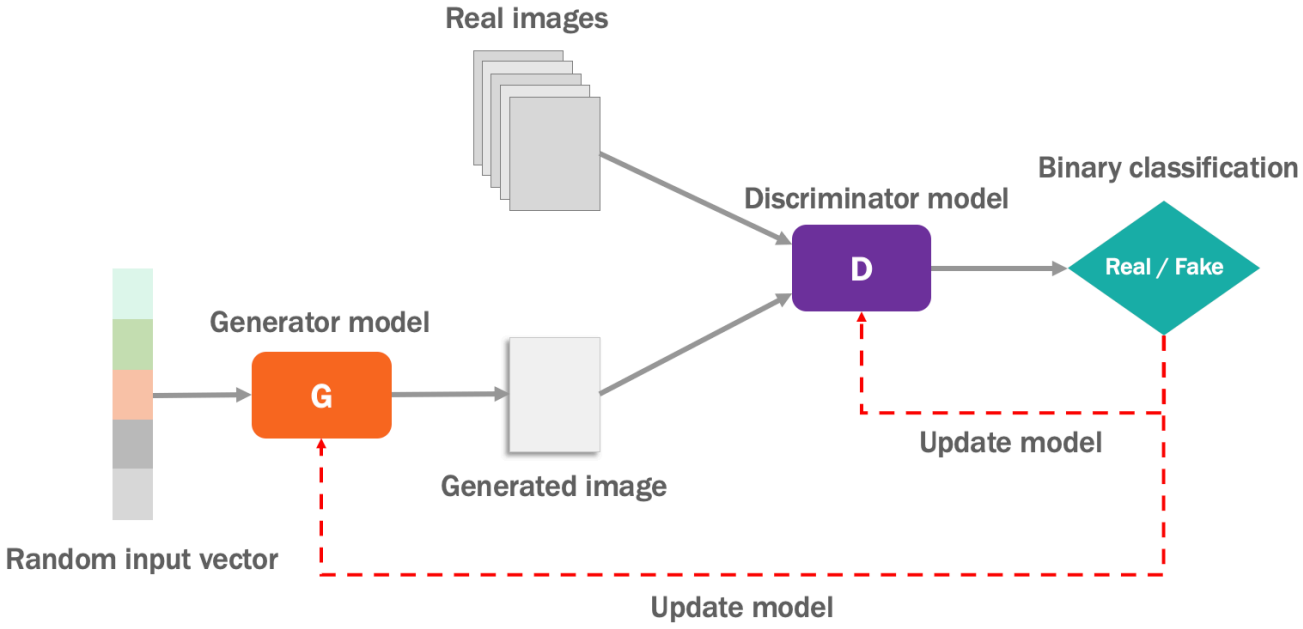
$$\underbrace{\mathbb{E}_{Q(z|x)}\log P(x|z)}_{\textbf{Maximize}} - \underbrace{D_{KL}\left(Q(z|x)||P(z)\right)}_{\textbf{Minimize}}$$

- reparameterization



9 Generative Adversarial Networks (GANs)

Architecture of GAN



Main components

Mini-max Game

- Generator and discriminator have “opposite” targets
- We’ll start with the discriminator which is a classifier
- The discriminator aims to distinguish between real and fake images

Cost funtion

GANs can be considered as a zero sum game or minmax problem with the cost function:

$$\theta_G^* = \arg\min_{\theta_G} \max_{\theta_D} -J_D$$

- discriminator cost function

$$J_D = -\left(\sum_x \log D(x) + \sum_z \log(1 - D(G(z)))\right)$$

- generator cost function

$$J_G = \sum_z \log(1 - D(G(z)))$$