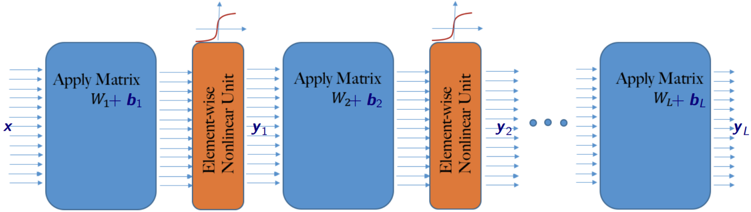
IFI 9000 Analytics Methods
More on Deep Learning and Generative Adversarial Networks

by **Houping Xiao**

Spring 2021

Quick Review: Architecture of Neural Networks



- A neural network consists of a sequence of multi-output linear units followed by nonlinear activations

$$\boldsymbol{y}_1 = \sigma_1(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1)$$

$$\boldsymbol{y}_2 = \sigma_2(\boldsymbol{W}_2\boldsymbol{y}_1 + \boldsymbol{b}_2)$$

$$\vdots$$

$$\boldsymbol{y}_L = \sigma_L(\boldsymbol{W}_L\boldsymbol{y}_{L-1} + \boldsymbol{b}_L)$$

Quick Review: Gradient Descent

- Recall when we had N training samples $(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \cdots, (\boldsymbol{x}^{(N)}, \boldsymbol{y}^{(N)})$ our fitting objective was in one of the forms:

$$\underset{\boldsymbol{p}}{min} \quad \frac{1}{N}\sum_{n=1}^{N} ||\boldsymbol{y}^{(n)} - \mathcal{M}_{\boldsymbol{p}}\left(\boldsymbol{x}^{(n)}\right)||^2; \quad \underset{\boldsymbol{p}}{min} \quad \frac{1}{N}\sum_{n=1}^{N} \mathcal{H}\left(\boldsymbol{y}^{(n)}, \mathcal{M}_{\boldsymbol{p}}\left(\boldsymbol{x}^{(n)}\right)\right)$$

  Here $\boldsymbol{p}$ is the hyper parameter set: $\boldsymbol{W}_1, \cdots, \boldsymbol{W}_L, \boldsymbol{b}_1, \cdots, \boldsymbol{b}_L$

- As a result:

$$\mathcal{C}(\boldsymbol{p}) = \frac{1}{N}\mathcal{C}_n(\boldsymbol{p}) \rightarrow \bigtriangledown\mathcal{C}(\boldsymbol{p}) = \frac{1}{N} \bigtriangledown \mathcal{C}_n(\boldsymbol{p})$$

- Gradient descent with **learning rate** η and **momentum** γ:

$$\boldsymbol{\theta}_{k+1} = \gamma\boldsymbol{\theta}_k + \eta \bigtriangledown \mathcal{C}(\boldsymbol{p}^k)$$

$$\boldsymbol{p}^{k+1} = \boldsymbol{p}^k - \boldsymbol{\theta}_{k+1}$$

Back propagation

- This is another terminology that you probably hear a lot in deep learning
- Recall that you had to calculate the derivative with respect to each sample and each sample function is a complicated nested function, e.g.,

$$C_n = \left|\left|\boldsymbol{y}^{(n)} - f_L(f_{L-1}(f_{L-2}(_1(\boldsymbol{x})\cdots)))\right|\right|^2, \quad , f_l(\boldsymbol{z}) = \sigma_l(\boldsymbol{W}_l\boldsymbol{z} + \boldsymbol{b}_l)$$

- Back propagation is simply the application of the chain rule to calculate the derivative of nested functions like C_n in terms of all the unknown parameters $\boldsymbol{W}_1, \cdots, \boldsymbol{W}_L, \boldsymbol{b}_1, \cdots, \boldsymbol{b}_L$
- Since the actual story goes through a lot of indexing complications, let me explain things via a simple example

Back propagation, chain rule simple example

- Find the derivative of the following function at $w = 2$:

$$f(w) = (\sin(w^2 + 1))^2$$

- Solution: Notice that

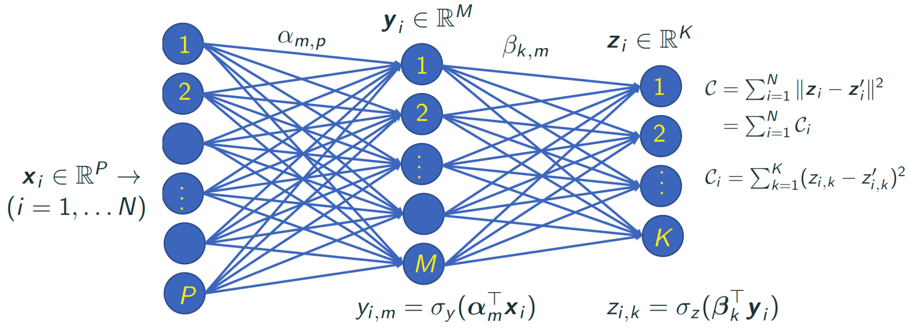
$$f = g_1(g_2(g_3(w))); \quad g_1(g_2) = g_2, g_2(g_3) = \sin(g_3), g_3(w) = w^2 + 1$$

and use the chain rule

$$\frac{\partial f}{\partial w} = \frac{\partial g_1}{\partial w} = \frac{\partial g_1}{\partial g_2}\frac{\partial g_2}{\partial g_3}\frac{\partial g_3}{\partial w} = 2\sin(5) \times \cos(5) \times 4$$

- Some useful videos about back propagation:
  - https://www.youtube.com/watch?v=Ilg3gGewQ5U
  - https://www.youtube.com/watch?v=tIeHLnjs5U8

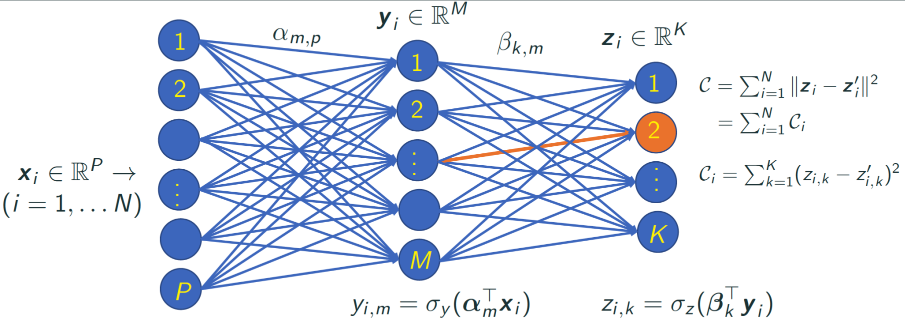
Back propagation



- Use chain rule to derive

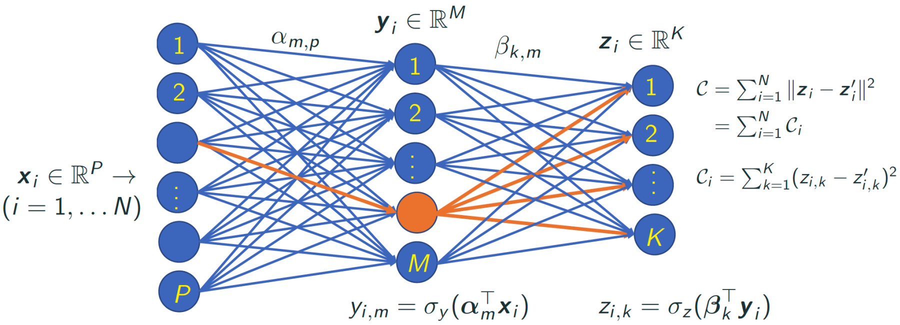
$$\frac{\partial \mathcal{C}_i}{\partial \beta_{k_0,m_0}}, \frac{\partial \mathcal{C}_i}{\partial \alpha_{m_0,p_0}}$$

Back propagation



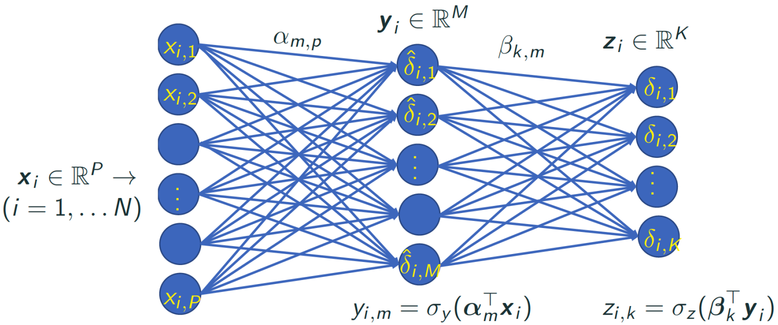
- Last layer sensitivity:

$$\begin{aligned}
\frac{\partial \mathcal{C}_i}{\partial \beta_{k_0,m_0}} &= \frac{\partial \mathcal{C}_i}{\partial z_{i,k_0}} \frac{\partial z_{i,k_0}}{\partial \beta_{k_0,m_0}} \\
&= 2(\sigma_z(\beta_{k_0}^\top \boldsymbol{y}_i) - z'_{i,k_0})\sigma'_z(\beta_{k_0}^\top \boldsymbol{y}_i) y_{i,m_0} \\
&= \delta_{i,k_0} y_{i,m_0}
\end{aligned}$$



- Other layers sensitivity:

$$
\begin{aligned}
\frac{\partial \mathcal{C}_i}{\partial \alpha k_0, m_0} &= \sum_{k=1}^K \frac{\partial \mathcal{C}_i}{\partial z_{i,k}} \frac{\partial z_{i,k}}{\partial y_{i,m_0}} \frac{\partial y_{i,m_0}}{\partial \alpha_{k_0,m_0}} \\
&= \sum_{k=1}^K 2(\sigma_z(\beta_k^\top \boldsymbol{y}_i) - z'_{i,k})\sigma'_z(\beta_k^\top \boldsymbol{y}_i)\beta_{k,m_0}\sigma'_y(\alpha_{m_0}^\top \boldsymbol{x}_i)x_{i,p_0} \\
&= \sigma'_y(\alpha_{m_0}^\top \boldsymbol{x}_i)\left(\sum_{k=1}^K \delta_{i,k}\beta_{k,m_0}\right) x_{i,p_0} = \hat{\delta}_{i,m_0} x_{i,p_0}
\end{aligned}
$$

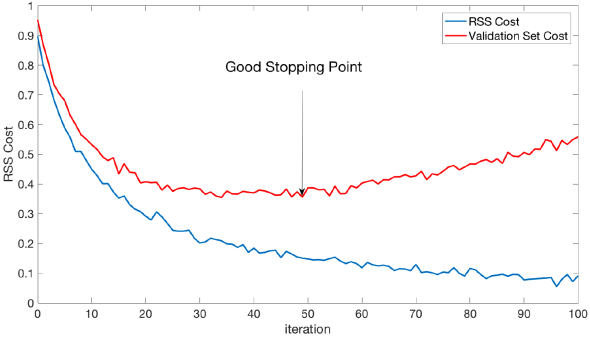


- Sensitivity summary:

$$\frac{\partial \mathcal{C}_i}{\partial \beta_{k_0,m_0}} = \delta_{i,k_0} y_{i,m_0}, \qquad \frac{\partial \mathcal{C}_i}{\partial \alpha_{m_0,p_0}} = \hat{\delta}_{i,m_0} x_{i,p_0}$$

Using a validation set to control the minimization

- As you observed in the previous slides gradient descent gradually decreases the RSS (or cross entropy cost) to find a minimizer
- One way to avoid over-fitting, is to use a "**validation set**", independent of the training set and stop the gradient descent iterations when the validation error starts to increase

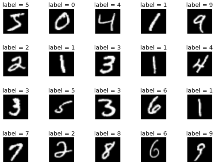


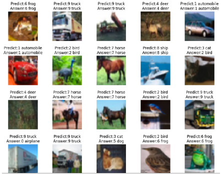
Regularization of neural networks to avoid overfitting

- Similar to linear models there are variety of techniques to avoid over-fitting in neural networks
  - L2 regularizers (similar to Ridge)
  - L1 regularizers (Similar to LASSO)
  - Dropout
    - See video: https://www.youtube.com/watch?v=ARq74QuavAo
    - See papers: Paper 1

Convolutional neural networks

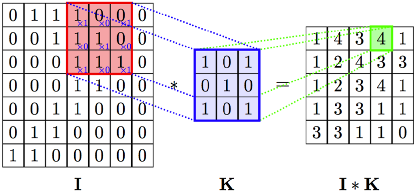
- Deep learning has shown a lot of promise in classifying images





Linear filtering and images

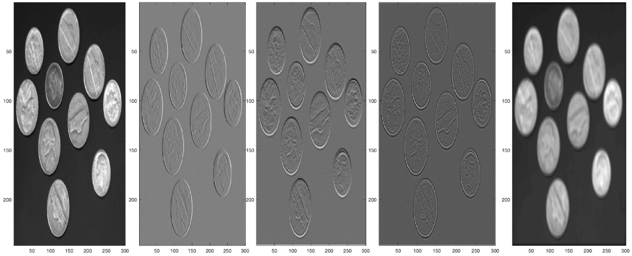
- Convolution is a linear operator widely used in image and signal processing



$$\boldsymbol{I} \star \boldsymbol{K}(m,n) = \sum_{i=1}^{M}\sum_{j=1}^{N} \boldsymbol{I}(m-i, n-j)\boldsymbol{K}(i,j)$$

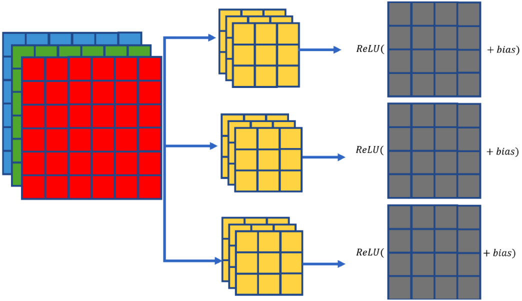
- Depending on the type of filter we pick for K the output image can have different properties (blurred, sharpened, edges detected, etc)

Examples of image convolution with different kernels



- If the filters are selected wisely, their output can be considered as alternative features to pixels
- In a CNN, we let the neural network learn these filters! In other words, CNN wisely chooses the right features that are the best for prediction
- For color images (RGB) we can have 3D filters each filter applicable to one channel

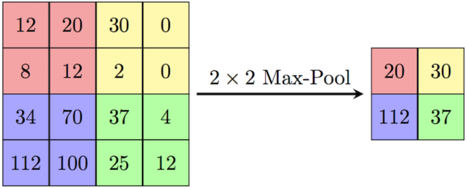
Convolutional layers



- We can define as many 2D or 3D convolutional filters (here 3 3D filters of size $3 \times 3 \times 3$)
- The total number of parameters that need to be learnt for this layer is going to be $3 \times (27 + 1)$
- An input image of $6 \times 6 \times 3$ is mapped to a tensor of size $4 \times 4 \times 3$

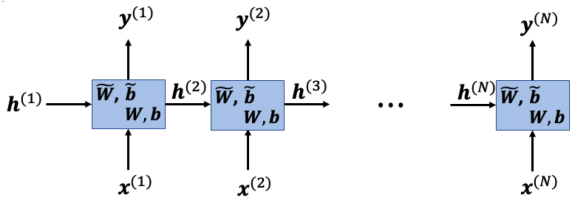
Max pooling

- Is another operation that allows us to reduce the input size by taking a max operation over smaller windows across the image



Recurrent neural networks

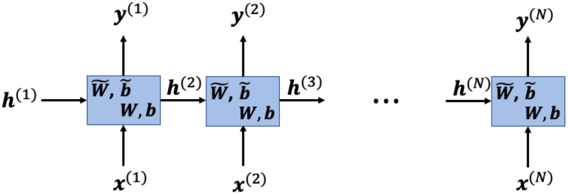
- While CNNs work quite promising for images, they may not be the best modeling tools for other data sets such as time series data
- For temporal, or time-series data and stream inputs (e.g., text streams), recurrent neural networks (RNNs) are of major attention



- We assume a sequence of data is streamed as N time instances, and mapped to a sequence of response (here of the same length).
- For now let's assume that the input and output have similar lengths

RNN: governing equations

- Remember in standard neural network the output of the hidden layer was in the form $\boldsymbol{h} = \sigma(\boldsymbol{Wx}x + \boldsymbol{b})$



- In RNNs the input is a stream x(t) and we have another coefficient matrix that makes the current hidden output dependent on the previous one:

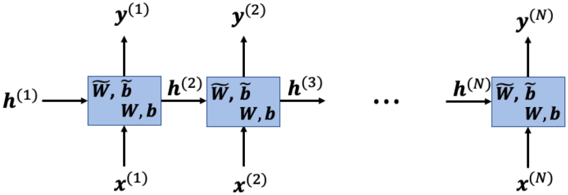
$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}\begin{pmatrix} \boldsymbol{h}^{(t-1)} \\ \boldsymbol{x}^{(t-1)} \end{pmatrix} + \boldsymbol{b}\right),$$

$$\boldsymbol{y}^{(t)} = \sigma\left(\tilde{\boldsymbol{W}}\boldsymbol{h}^{(t)} + \tilde{\boldsymbol{b}}\right), t = 1, \cdots, N$$

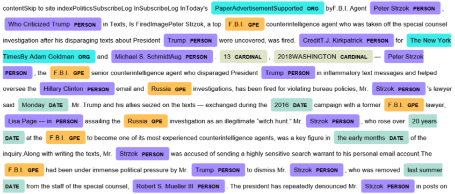
- Training cost per sample: $\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_{t=1}^{N} L\left(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}\right)$

Types of RNN and applications

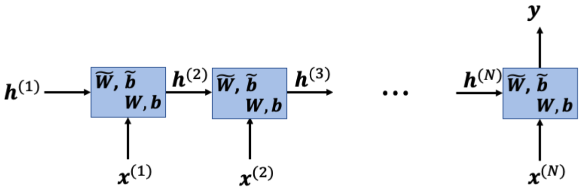
- The following architecture is many-to-many, with the input and output having the same length



- Application example is named-entity recognition (classify unstructured text into predefined classes)



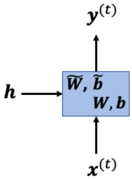
- The following architecture is many-to-one



- Application example is sentiment classification (review systems, scoring systems)

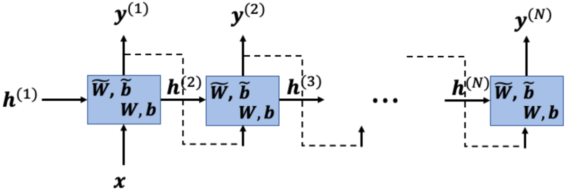
Types of RNN and applications

- The following architecture is one-to-one



- This is somehow equivalent to traditional one-layer network (real-time mapping)

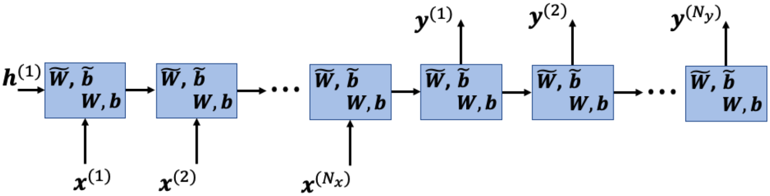
- The following architecture is one-to-many



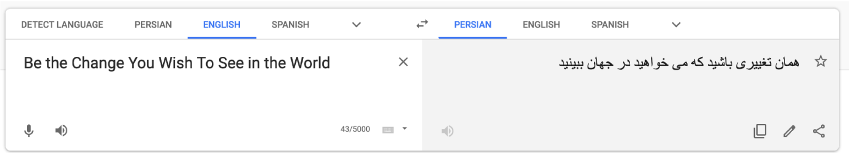
- Application example is music generation

Types of RNN and applications

- The following architecture is many-to-many, with the input and output having different lengths

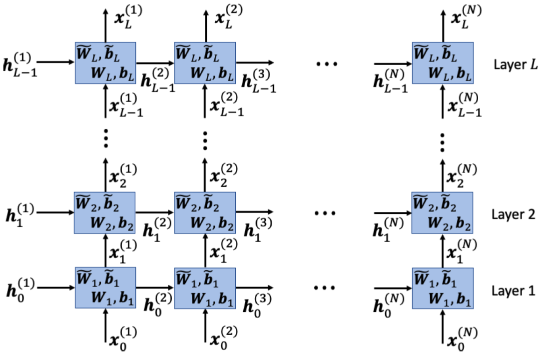


- Application example is machine translation



Deep RNNs

- All the architectures we explained so far can become deep and layered



- In practice we do not need very deep RNNs (unlike standard DNNs which can be very deep)

Deep RNNs

- One hot encoding is normally used to convert a vocabulary into digital inputs

	Me	Go	Hunting
1. Me	1	0	0
2. Water	0	0	0
3. Food	0	0	0
4. Cave	0	0	0
5. Go	0	1	0
6. Dinosaur	0	0	0
7. Sleep	0	0	0
8. Stone	0	0	0
9. Hunting	0	0	1
10. Stick	0	0	0

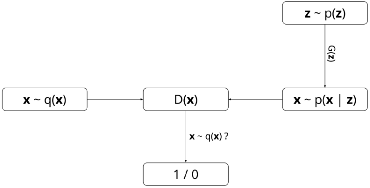
- It is normally easier and more robust to do the one hot encoding with the words other than letters

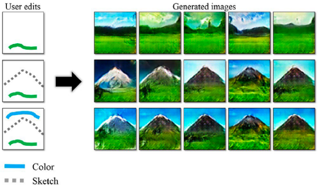
Problems with standard RNNs and remedies

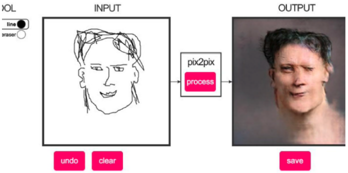
- Hard to train and vanishing gradient
- Difficulty accessing information from long time ago
- Two main variants of RNNs:
  - Long Short-Term Memory (LSTM)
  - Gated Recurrent Units (GRUs)
- To learn more and see some cool applications see: https://www.youtube.com/watch?v=6niqTuYFZLQt=1850s

Deep RNNs

- Is the most recent breakthrough in machine learning started in 2015
- Basically once we pass enough samples to a GAN network, it starts to learn how to generate similar samples







- To learn more and see some interesting applications see: This Video, or This Video

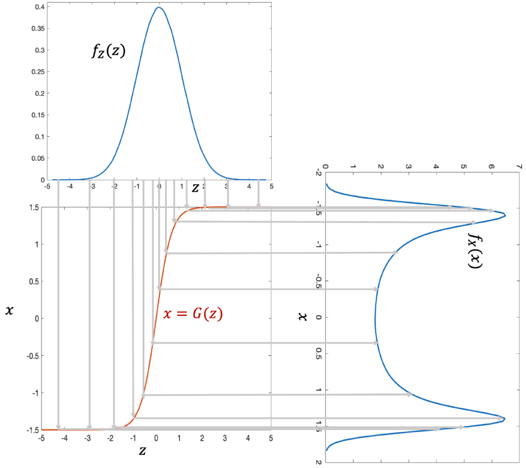
Generative Adversarial Networks

Introduction to GANs

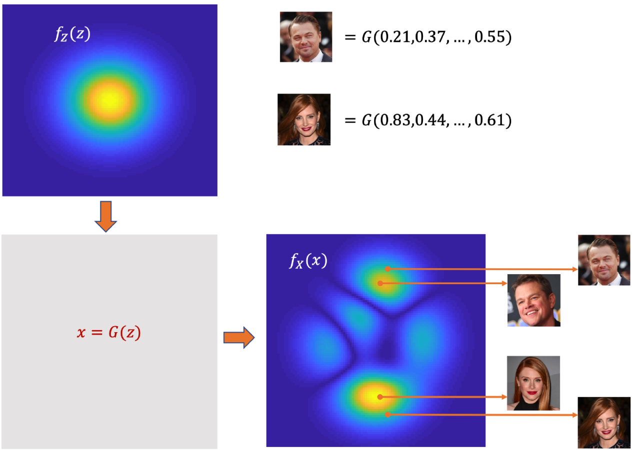
- GAN is an unsupervised learning technique which allows us to **model complex distributions** and sample from them
- Examples of these complex distributions are the space of **natural images**, such as people's images
- Intuitively, GANs train a neural network in an "**adversarial way**" to map a simple distribution to the target complex distribution

Basics of GANs

- Simple distributions such as standard (multivariate) normal can be mapped to more complex distributions once passed through a function

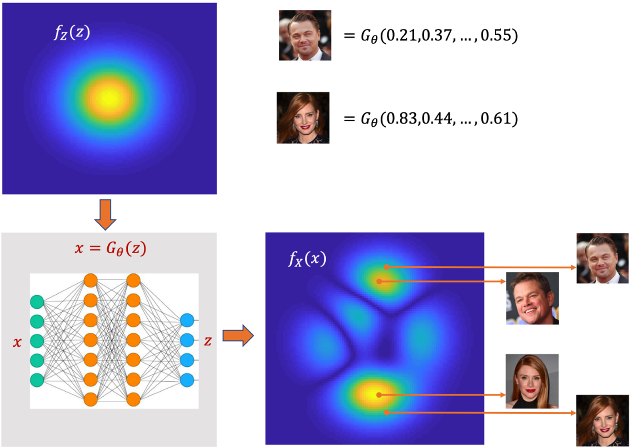


- This trick can be applied to complex distributions such as space of natural images (e.g., celebrities)



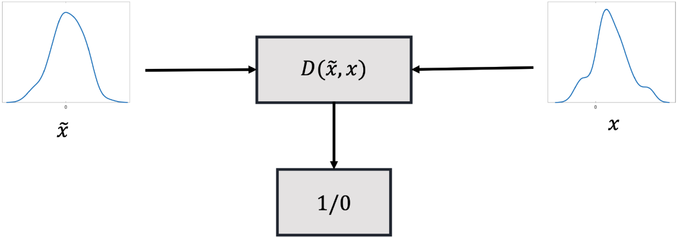
Basics of GANs

- Can we train a neural network $G_\theta(z)$ that learns to perform this mapping? (this is essentially what GANs do)



Basics of GANs

- GANs do this task in an adversarial way
- To understand this better, let's start with a quick overview of logistic regression



Basics of GANs: logistic regression overview

- In binary logistic regression, we have a set of training samples $(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_N, y_N)$, where $\boldsymbol{x}_i \in \mathbb{R}^p$ and $y_i \in \{0, 1\}$
- In logistic regression we assume

$$p(\boldsymbol{x}_i) = sigmoid(\boldsymbol{w}\top\boldsymbol{x}_i) = \frac{\exp(\boldsymbol{w}\top\boldsymbol{x}_i)}{1 + \exp(\boldsymbol{w}\top\boldsymbol{x}_i)} = \mathbb{P}(y = 1|\boldsymbol{x}_i) = 1 - \mathbb{P}(y = 0|\boldsymbol{x}_i)$$

We aim to maximize the MLE cost:

$$\begin{aligned}\mathbb{P}(Y_1 = y_1, \cdots, Y_N = y_n|\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N, \boldsymbol{w}) &= \prod_{i=1}^N \mathbb{P}(Y_i = y_i|\boldsymbol{x}_i, \boldsymbol{w}) \\ &= \prod_{i:y_i=1} p(\boldsymbol{x}_i) \prod_{i:y_i=0}(1 - p(\boldsymbol{x}_i)) \\ &= \prod_{i=1}^N p(\boldsymbol{x}_i)^{y_i}(1 - p(\boldsymbol{x}_i))^{1-y_i}\end{aligned}$$

- After applying log, and normalization, we aim to maximize

$$\frac{1}{N}\sum_{i=1}^N y_i \log(p(\boldsymbol{x}_i)) + (1 - y_i)\log(1 - p(\boldsymbol{x}_i))$$

Basics of GANs: logistic regression overview

- In a nutshell, in logistic regression we assume

$$p(\boldsymbol{x}_i) = sigmoid(\boldsymbol{w}^\top \boldsymbol{x}_i)$$

and end up maximizing the objective

$$\frac{1}{N}\sum_{i=1}^{N} y_i \log(p(\boldsymbol{x}_i)) + (1 - y_i)\log(1 - p(\boldsymbol{x}_i))$$

- In order to make our classifier stronger, we can use a DNN for $p(\boldsymbol{x}_i)$, i.e.,

$$p(\boldsymbol{x}_i) = D_\theta(\boldsymbol{x}_i)$$

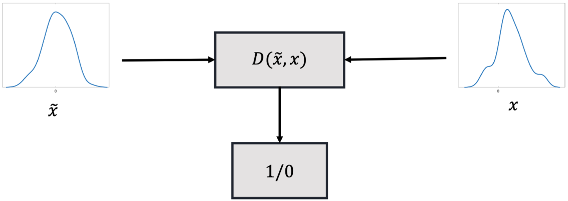
and end up doing the maximization

$$\frac{1}{N}\sum_{i=1}^{N} y_i \log(D_\theta(\boldsymbol{x}_i)) + (1 - y_i)\log(1 - D_\theta(\boldsymbol{x}_i))$$

- We ended up with

$$\frac{1}{N}\sum_{i=1}^{N} y_i \log(D_\theta(\boldsymbol{x}_i)) + (1 - y_i)\log(1 - D_\theta(\boldsymbol{x}_i))$$

- Considering $\boldsymbol{x}_i$ the samples with label 1, and $\tilde{\boldsymbol{x}}_i$, the samples with label 0,



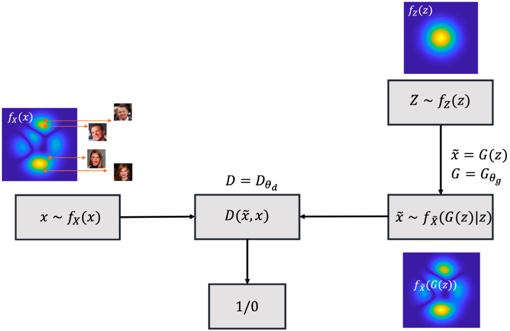
in a more closed-form representation we have:

$$\max_{\boldsymbol{\theta}} \quad \mathbb{E}_{\boldsymbol{x}} \log(D_\theta(\boldsymbol{x})) + \mathbb{E}_{\tilde{\boldsymbol{x}}} \log(1 - D_\theta(\tilde{\boldsymbol{x}}))$$

Basics of GANs

- We would like $\tilde{\boldsymbol{x}}$ to be generated by $G(\boldsymbol{z})$, so we setup a competition between Generator and Discremenator

$$\min_{\boldsymbol{\theta}_g} \max_{\boldsymbol{\theta}_d} \quad \mathbb{E}_{\boldsymbol{x}} \log(D_{\theta_d}(\boldsymbol{x})) + \mathbb{E}_{\tilde{\boldsymbol{x}}} \log(1 - D_{\theta_d}(G_{\theta_g}(\boldsymbol{z}))$$



- Generator tries to fool the discremenator, and discremenator tries to identify "fake" samples

- The optimization goes through several gradient ascent steps to update the discremenator, followed by a gradient descent step to update the generator

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
   • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
   • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
   • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

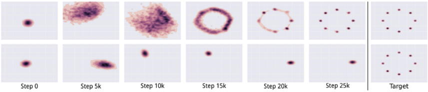
  end for
  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  • Update the generator by descending its stochastic gradient:

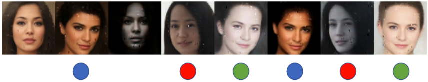
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Well-known issues with GANs

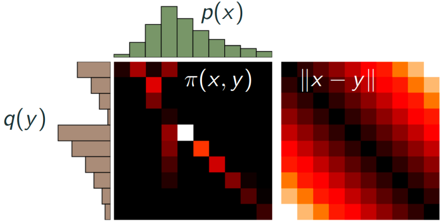
- Watch **this video** to find out what GANs can do
- Some well-known problems with GANs
  - divergence
  - vanishing gradient
  - unstable gradient
  - mode collapse





Wasserstein GANs

- Is a more robust scheme compared to conventional GANs, derived based on the sitance between distributions



Source: https://vincentherrmann.github.io/images/wasserstein/transport_plan.png

$$D_W(p, q) = \inf_{\pi \in \Pi(p,q)} \mathbb{E}_{(x,y)\sim\pi} ||x - y||$$

- $\Pi(p, q)$ is the space of all joint distributions with marginals p and q
- In discrete case, this program can be written as an LP $(\pi(x_i, y_j) \to \pi_{i,j}, c_{i,j} = ||x_i - y_j||$

$$\min_{\pi_{i,j} \geq 0} \sum_{i,j} c_{i,j}\pi_{i,j} \quad \text{s.t.} \quad \sum_i \pi_{i,j} = q_j, \sum_j \pi_{i,j} = p_i$$

Wasserstein GANs

- In discrete case, this program can be written as an LP $(\pi(x_i, y_j) \to \pi_{i,j}, c_{i,j} = ||x_i - y_j||$

$$\min_{\pi_{i,j} \geq 0} \sum_{i,j} c_{i,j} \pi_{i,j} \quad \text{s.t.} \quad \sum_i \pi_{i,j} = q_j, \sum_j \pi_{i,j} = p_i$$

- Using LP duality we can rewrite $D_w(p, q)$ as

$$D_w(p, q) = \max_{\gamma_i, \lambda_j} \sum_i \gamma_i p_i + \sum_j \lambda_j q_j \quad \text{s.t.} \gamma_i + \lambda_j \leq c_{ij}$$

- When c is a proper distance, we must have $\lambda = -\gamma$ and therefore in continuum:

$$D_w(p, q) = \sup_{||f||_L \leq 1} \mathbb{E}_{x \sim p} f(x) - \mathbb{E}_{y \sim p} f(y)$$

- 1-Lipschitz function: $||f||_L \leq 1$ equivalent to $|f(x) - f(y)| \leq ||x - y||$

Wasserstein GANs

- We showed that the following maximization over the space of 1-Lipschitz functions gives the distance between two distributions

$$D_w(p,q) = \sup_{||f||_L \leq 1} \mathbb{E}_{x\sim p} f(x) - \mathbb{E}_{y\sim p} f(y)$$

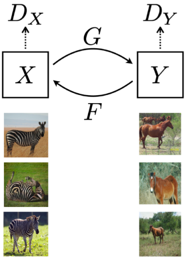
- So we can conveniently modify this to a min-max game to formulate the so-called Wasserstein GANs:

$$\min_G \max_{||f||_L \leq 1} \quad \mathbb{E}_X f(x) - \mathbb{E}_Z f(G(z))$$

- Here, f and G can be deep neural networks and the min-max can be performed over their underlied parameters

Cycle GANs

- Instead of mapping a simple distribution to a complex one, we map a complex to another complex and enforce a cycle consistency



$$\mathcal{L}(G, D_Y) = \mathbb{E}_{\mathbf{Y}} \log(D_Y(\mathbf{y})) + \mathbb{E}_{\mathbf{X}} \log(1 - D_Y(G(\mathbf{x})))$$

$$\mathcal{L}(F, D_X) = \mathbb{E}_{\mathbf{X}} \log(D_X(\mathbf{x})) + \mathbb{E}_{\mathbf{Y}} \log(1 - D_X(F(\mathbf{y})))$$

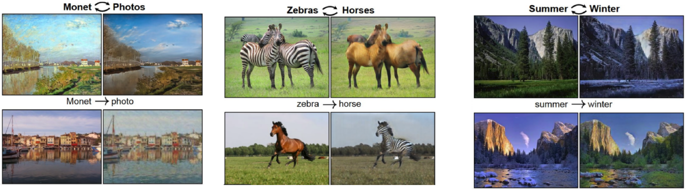
$$\mathcal{L}_{cyc}(F, G) = \mathbb{E}_{\mathbf{X}} \|F(G(\mathbf{x})) - \mathbf{x}\| + \mathbb{E}_{\mathbf{Y}} \|G(F(\mathbf{y})) - \mathbf{y}\|$$

- We solve the following min-max game

$$\min_{F,G} \max_{D_X, D_Y} \mathcal{L}(G, D_Y) + \mathcal{L}(F, D_X) + \mathcal{L}_{cyc}(F, G)$$

Cycle GANs

- See **this video** as an example of what can be done with cycle-GANs



Failure Case

The End