Perceptron Learning

```
init: $w^0 = 0$
while $\exists i_k : w^{k^T} x^{i_k} y^{i_k} < 0$ do
    $w^{k+1} \leftarrow w^k + \eta(y - \hat{y})x^{i_k}$
    where $\hat{y} := w^T x \geq 0$
end while
```

Activation Functions

Sigmoid

$\sigma(\eta) = sigm(\eta) = \frac{1}{1+e^{-\eta}} = \frac{e^\eta}{e^\eta+1}$
$\frac{\partial}{\partial\eta}\sigma(\eta) = \sigma(\eta)(1-\sigma(\eta))$

ReLU

$g(x) = max\{0, z\}$

Tanh

$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
$\frac{\partial}{\partial x}tanh(x) = 1 - tanh^2(x)$

MLE

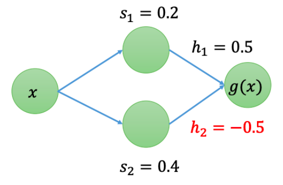
LS and Cross entropy are MLE estimators. Direct link to KL divergence.

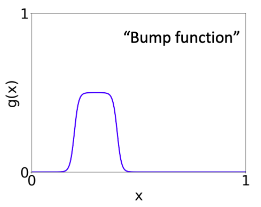
Cross Entropy as MLE

Logistic Regression MLE:
$y^{(i)} \sim Bernoulli(\sigma(w^T x_i))$
$P(D|w) = \prod_i (\frac{1}{1+e^{-w^T x_i}})^{y_i}(1 - \frac{1}{1+e^{-w^T x_i}})^{1-y_i}$
$-log(P(D|w)) =$
$-\frac{1}{n}\sum y_i log(\pi_i) + (1-y_i)log(1-\pi_i)$ (Cross Entropy)

Universal approximation theorem

A feed-forward neural network with a single hidden layer and continuous non-linear activation function can approximate any continuous function with arbitrary precision. $\exists g(x)$ as NN , $g(x) \approx f(x) = \sum v_i \sigma(w_i^T x_i + b_i), |f(x) - g(x)| < \epsilon$. How?Creating step function with $w^T + b$ and σ function. Increase w to create the step, where is the location? $s := -\frac{b}{w}$

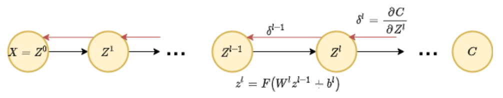




Backpropagation

Multivariate Chain Rule

$$\text{Univariate: } \frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$
$$\text{Multivariate: } \frac{\partial z_j}{\partial x_i} = \sum_m \frac{\partial z_j}{y_m}\frac{\partial y_m}{\partial x_i}$$



$X = Z^0$
Z^1
...
δ^{l-1}
$\delta^l = \frac{\partial C}{\partial Z^l}$
Z^{l-1}
Z^l
...
C
$z^l = F(W^l z^{l-1} + b^l)$

$$a^l := W^l z^{l-1} + b^l \Rightarrow z^l = F(a^l)$$
$$\frac{\partial C}{\partial z^l} = \delta^{l+1}\frac{\partial z^{l+1}}{z^l}$$
$$\frac{\partial C}{\partial W^l_{ij}} = \delta^l \frac{\partial z^l}{\partial W^l_{ij}} = vec_i^{n_l} F'(a_i^l) z_j^{l-1}$$
$$\frac{\partial C}{\partial b_i^l} = \delta^l vec_i^{n_l} F'(a_i^l)$$
$$\frac{C}{\partial W^l} = \sum_i \delta_i^l \frac{\partial z_i^l}{\partial W^l} = \sum_i \delta_i^l [n_l * n_{l-1} \text{ matrix}$$

with elements on ith row and zero elsewhere]

HMAX Model

A model inspired by neuroscience and similar to CNN.

- S-cell response:
  $y = exp(-\frac{1}{2\sigma^2}\sum_1^{n_{s\,k}}(w_j - x_j)^2)$
- C-cell response: $y = \underset{j=1\ldots n_{C_k}}{x_j} \quad max$

Transformers

- Transform T is linear if:
  $T(\alpha u + \beta v) = \alpha T(u) + \beta T(v)$
- A transform T is invariant to f if:
  $T(f(u)) = T(u)$
- A transform T is equivariant to f if:
  $T(f(u)) = f(T(u))$
- Linear Filtering: $I'(i,j) = \sum_{(m,n)\in N(i,j)} K(i,j,m,n)I(i+m, j+n)$ if K doesn't depend on i, j then the transformation is shift-invariant.

CNN

Convolution and Correlation

Correlation:

$$I'(i,j) = \sum_{m=-k}^{m=k}\sum_{n=-k}^{n=k} K(m,n)I(i+m, j+n)$$

Convolution:

$$I'(i,j) = \sum_{m=-k}^{m=k}\sum_{n=-k}^{n=k} K(m,n)I(i-m, j-n)$$

K(-1,-1)	K(0, -1)	K(1, -1)
K(-1, 0)	K(0, 0)	K(1, 0)
K(-1, 1)	K(0, 1)	K(1, 1)

$$I'(x,y) = K(1,1)I(x-1,y-1) + K(0,1)I(x,y-1) + K(-1,1)I(x+1,y-1) + K(1,0)I(x-1,y) + K(0,0)I(x,y) + K(-1,0)I(x+1,y) + K(1,-1)I(x-1,y+1) + K(0,-1)I(x-1,y) + K(-1,-1)I(x+1,y+1)$$

If $K(i,j) = K(-i,-j)$ correlation and convolution are equal. Can be implemented as multiplying $(m+n-1)*n$ matrix to the vectorized input.

CNN receptive field and parameter number

$r_{out} = r_{in} + (k-1) * j_{in}$ where k is filter size and j_{in} is jump which is equal to stride in the layer. number of parameters is ((shape of width of the filter * shape of height of the filter * number of filters in the previous layer+1)*number of filters)

CNN-Backward Pass

$$\delta_{ij}^{l-1} = \frac{\partial C}{\partial z_{ij}^{l-1}} = \sum_{i'}\sum_{j'}\frac{\partial C}{\partial z^l_{i'j'}}\frac{\partial z^l_{i'j'}}{\partial z^{l-1}_{ij}}$$
$$= \sum_{i'}\sum_{j'}\delta^l_{i'j'}\frac{\partial}{\partial z^{l-1}_{ij}}\sum_m\sum_n(W^l_{mn}z^{l-1}_{i'-m,j'-m}) + b$$
$$= \sum_{i'}\sum_{j'}\delta^l_{i'j'}W^l_{mn}\ i = i' - m, j = j' - n$$
$$= \sum_{i'}\sum_{j'}\delta^l_{i'j'}W^l_{i'-i,j'-j}$$
$$= \sigma^l * ROT_{180}(W^l) = \text{cross\_corr}(\sigma^l, W^l)$$

sliding W^l over σ^l

CNN-Parameter Update

$$\frac{\partial C}{\partial W^l_{m,n}} = \sum_i\sum_j\frac{\partial C}{\partial z^l_{ij}}\frac{\partial z^l_{ij}}{\partial W^l_{m,n}}$$
$$= \sum_i\sum_j\delta^l_{ij}\frac{\partial}{\partial W^l_{m,n}}\sum_m\sum_n(W^l_{mn}z^{l-1}_{i-m,j-m}) + b$$
$$= \sum_i\sum_j\delta^l_{ij}z^{l-1}_{i-m,j-m}$$
$$= \sigma^l * ROT_{180}(z^{l-1}) = \text{cross\_corr}(\sigma^l, z^{l-1})$$

sliding W^l over z^{l-1}

Max Pooling

Forward Pass: $z^l = max\{z_i^l\}$
Backward Pass:
$$\frac{\partial z_k^l}{\partial z^{l-1}} = \begin{cases}1 & \text{if } k = argmax\ \ z_k^{l-1}\\ 0 & o.w\end{cases}$$

Image Classification Models Comparison

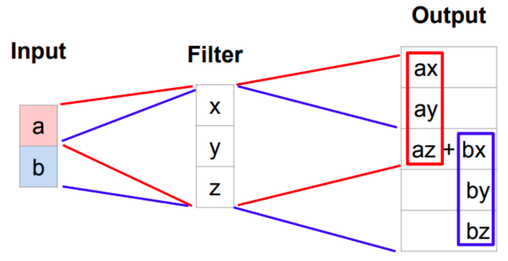
VGG has more layers and larger receptive field, but less parameters(smaller filters). GoogleNet has even more layers and larger recpetive field. It has no FC layer. It has inception module(using 1x1 convolutions to reduce channel dimentionality) and less parameters. ResNet has many many more layers. problem with more layers? not overfitting but it's hard to optimize. solution? add identity connections and just learn the residuals which is what the ResNet do.

Object Detection

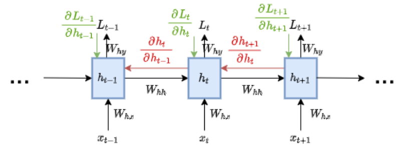
First extract region proposals(using segmentation based on intensity, etc.), give these regions to CNN, give CNN's results to SVM to decide weather that region contains an object or not and also refine the region.

Upsampling-Transposed Convolution

In what task? Pixel-wise classification of images. *In what model?* **Fully convolutional neural networks**. First increase the dimentionality and then upsample. Instead of $Kx = y$ use $K^T y = x$ to upsample.



RNN


$\frac{\partial L_{t-1}}{\partial h_{t-1}}$
L_{t-1}
$\frac{\partial L_t}{\partial h_t}$
L_t
$\frac{\partial L_{t+1}}{\partial h_{t+1}}$
L_{t+1}
W_{hy}
$\frac{\partial h_t}{\partial h_{t-1}}$
$\frac{\partial h_{t+1}}{\partial h_t}$
h_{t-1}
h_t
h_{t+1}
W_{hh}
W_{hx}
x_{t-1}
x_t
x_{t+1}

$h_t = f(h_{t-1}, x_t, W),\ \hat{y}_t = W_{hy}h_t,\ L_t = \|y_t - \hat{y}_t\|^2$

$h_t = tanh(W_{hh}h_{t-1} + W_{hx}x_t + b)$

but Assume: $h_t = W_{hh}f(h_{t-1}) + W_{hx}x_t$

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{s}\frac{\partial L_t}{\partial W}\ ; \frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\sum_{k=1}^{t}\frac{\partial h_t}{\partial h_k}\frac{\partial^+ h_k}{\partial W}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t}\frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} W_{hh}^T\, diag[f'(h_{i-1})]$$

$$\Rightarrow \forall i : \|\frac{\partial h_i}{\partial h_{i-1}}\| \le \|W_{hh}^T\|\|diag[f'(h_{i-1})]\|$$

$$\le \|W_{hh}^T\|\gamma$$

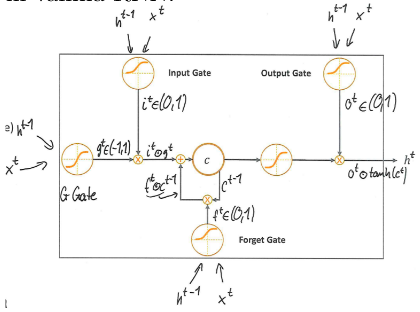
if $\|W_{hh}^T\| = \lambda < \frac{1}{\gamma}$

$\Rightarrow \|\frac{\partial h_i}{\partial h_{i-1}}\| \le \eta < 1 \Rightarrow \|\frac{\partial h_t}{\partial h_k}\| < \eta^{t-k}$ vanish

if $\lambda > \frac{1}{\gamma}$ explode

LSTM

We have additive behaviour to the cell state instead of multiplicative behaviour that we had in vanilla RNN.



$$c^t = f^t \odot c^{t-1} + i^t \odot g^t$$

$$h^t = o^t \odot tanh(c^t)$$

Generative Models

Taxonomy

Generative Models				
Implicit Density		Explicit Density		
Markov Chain	Direct	Tractable	Approximate	
			Markov Chain	Variational
GSN	**GAN**	**Belief Nets**		
-	-	**NADE**	**Variational Autoencoders**	*Boltzmann Machine*
-	-	**MADE**	-	-
-	-	**PixelRNN/CNN**	-	-

Auto Encoders

Auto encoder with linear f, g is equal to PCA. $\hat{\theta}_f, \hat{\theta}_g = \underset{\theta_f,\theta_g}{argmin} \sum_n^N \|x_n - g(f(x_n))\|^2$

By increasing the latent space dimension, Auto encoder generates better samples but performance of PCA doesn't quite change.

Kullback-Leibler divergence

$D_{KL}(P||Q) = \int_x p(x)log\frac{p(x)}{q(x)}dx$

KL is not symmetric. **KL is always positive:**

$$-D_{KL} = \int_x p(x)log\frac{q(x)}{p(x)}dx$$

$$\le^* log\int_x p(x)\frac{q(x)}{p(x)}dx = 0$$

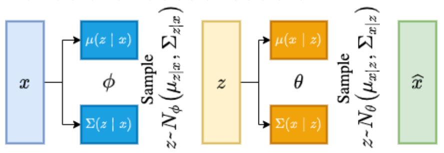
$*$:= Jensen for concave functions.

JS divergence

Unlike KL divergence this is a symmetric measure.

$D_{JS}(P||Q) = \frac{1}{2}D_{KL}(p||\frac{p+q}{2}) + \frac{1}{2}D_{KL}(q||\frac{p+q}{2})$

Variational autoencoders



Loss Function

Intractability: $p_\theta(x) = \int_z p_\theta(x|z)p_\theta(z)dz$ is interactable because of continuous integration. $p_\theta(z|x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)}$ is also interactable.

$$log\ p_\theta(x^{(i)}) = \mathbb{E}_{z\sim q_\phi(z|x^{(i)})}[log\ p_\theta(x^{(i)})]$$

$$= \mathbb{E}_{z\sim q_\phi(z|x^{(i)})}[log\frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(z|x^{(i)})}]$$

$$= \mathbb{E}_{z\sim q_\phi(z|x^{(i)})}[log\frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(z|x^{(i)})}\frac{q_\phi(z|x^{(i)})}{q_\phi(z|x^{(i)})}]$$

$$= \mathbb{E}_z[log\ p_\theta(x^{(i)}|z)] + \mathbb{E}_z[log\frac{p_\theta(z)}{q_\phi(z|x^{(i)})}]$$

$$+ \mathbb{E}_z[log\frac{q_\phi(z|x^{(i)})}{p_\theta(x^{(i)}|z)}]$$

$$= \overset{\text{Reconstruction error}}{\boxed{\mathbb{E}_z[log\ p_\theta(x^{(i)}|z)]}} - \overset{\text{Make z similar to Gaussian}}{\boxed{D_{KL}(q_\phi(z|x^{(i)}), p_\theta(z))}}$$

$$+ \overset{\text{Interactable but positive}}{\boxed{D_{KL}(q_\phi(z|x^{(i)}), p_\theta(x^{(i)}|z))}} \ge \overset{\text{ELBO}}{\boxed{\mathcal{L}(x^{(i)}, \theta, \phi)}}$$

Backprop

Reparametrization Trick: substitute $z \sim \mathcal{N}(\mu, \Sigma)$ with $z = \mu + \epsilon\sigma,\ \epsilon \sim \mathcal{N}(0,1) \Rightarrow z = f(x, \theta, \epsilon)$

$$\nabla_{\theta,\phi}\ \mathcal{L}(x, \theta, \phi)$$

$$= \nabla_{\theta,\phi}\ \mathbb{E}_z[log\ p_\theta(x^{(i)}|z)] + \mathbb{E}_z[log\frac{p_\theta(z)}{q_\phi(z|x^{(i)})}]$$

$$= \nabla_{\theta,\phi}\ \mathbb{E}_z[log\frac{p_\theta(x, z)}{q_\phi(z|x)}]$$

$$= \nabla_{\theta,\phi}\ \mathbb{E}_{\epsilon\sim\mathcal{N}(0,1)}[log\frac{p_\theta(x, f(x,\theta,\phi,\epsilon))}{q_\phi(f(x,\theta,\phi,\epsilon)|x)}]$$

$$= \mathbb{E}_{\epsilon\sim\mathcal{N}(0,1)}[\nabla_{\theta,\phi}\ log\ \overset{\text{Deterministic NN :=w}}{\boxed{\frac{p_\theta(x, f(x,\theta,\phi,\epsilon))}{q_\phi(f(x,\theta,\phi,\epsilon)|x)}}}]$$

$$\approx \frac{1}{N}\sum_n^N \nabla_{\theta,\phi}\ w(x, \theta, \phi, \epsilon)$$

β-VAE

Goal: Learn disentangled representation without supervision. Approach: modification of the VAE, introducing an adjustable hyperparameter beta that balances latent channel capacity and independence constraints with reconstruction accuracy.

$$\underset{\theta,\phi}{max}\ \mathbb{E}_{x\sim\mathcal{D}}[\mathbb{E}_{z\sim q_\phi(z|x)}\ log\ p_\theta(x|z)]$$

subject to: $D_{KL}(q_\phi(z|x), p_\theta(z)) < \delta$

$\mathcal{L}(\theta, \phi, \beta) =$

$\mathbb{E}_{z\sim q_\phi(z|x)}\ log\ p_\theta(x|z) - \beta D_{KL}(q_\phi(z|x), p_\theta(z))$

GAN

Dataset Generation

Assuming G is fixed, N samples from original data: $x^n \sim p_d$, N synthetic samples $z^n \sim \mathcal{N}(0,1)$. Dataset would be: $D = \{(x^1, 1), ..., (x^n, 1), (G(z^1), 0), ..., (G(z^n), 0)\}$.

Objective

Considering D, set G fixed: $\mathcal{L}(D) = -\frac{1}{2N}(\sum_{i=1}^N y^i log(D(x^i)) + \sum_{i=1}^N(1 - y^i)log(1 - D(x^i)))$ which is equivalent to: $\frac{-1}{2}(\mathbb{E}_{x\sim p_d}[log(D(x))] + \mathbb{E}_{z\sim p_z}[log(1 - D(G(z)))])$. Define V:

$V(G, D) =$

$\mathbb{E}_{x\sim p_d}[log(D(x))] + \mathbb{E}_{z\sim p_z}[log(1 - D(G(z)))]$

$$G^* = \underset{G}{argmin}\ \underset{D}{max}\ V(G, D)$$

Equilibrium Point

equilibrium point wrt D: Let $D^* = \underset{D}{argmax}\ V(G, D)$, Proposition: if $p_d = p_g$ then $D^* = \frac{p_d}{p_d+p_g}$. Proof:

$$V(G, D) = \int_x p_d(x)log(D(x))dx +$$

$$\int_z p_z(z)log(1 - D(G(z)))dz$$

$$= \int_x p_d(x)log(D(x)) + p_g(x)log(1 - D(x))dx$$

Optimal D: $\forall a, b\{0,0\}$, $f(y) = a\ log(y) + b\ log(1-y)$: $f'(y) = 0 \Rightarrow \frac{a}{y} - \frac{b}{1-y} = 0 \Rightarrow y = \frac{a}{a+b} \Rightarrow D^* = \frac{p_d}{p_d+p_g}$

equilibrium point wtr G: Theorem: global min is achieved when $p_d = p_g$ and $\underset{G}{min}V(G, D^*) = -log(4)$

$$V(G, D^*) = \mathbb{E}_{x\sim p_d}[log\frac{p_d}{p_d+p_g}]$$

$$+ \mathbb{E}_{x\sim p_g}[log\ (1 - \frac{p_d}{p_d+p_g})]$$

$$= \mathbb{E}_{x\sim p_d}[log\frac{p_d}{p_d+p_g}\frac{2}{2}] + \mathbb{E}_{x\sim p_g}[log\frac{p_g}{p_d+p_g}\frac{2}{2}]$$

$$= -log(2) + \mathbb{E}_{x\sim p_d}[log\frac{2p_d}{p_d+p_g}]$$

$$-\ log(2) + \mathbb{E}_{x\sim p_g}[log\frac{2p_g}{p_d+p_g}]$$

$$= D_{KL}(p_d||\frac{p_d+p_g}{2}) + D_{KL}(p_g||\frac{p_d+p_g}{2})$$

$$-\ log(4)$$

$$= 2D_{JS}(p_d, p_g) - log(4)$$

Training in practice - issue

Issue 1: Alternate between: Gradient ascent on D: $\underset{\theta_d}{max}\ \mathbb{E}_{x\sim p(x)}[log\ D_{\theta_d}(x)] + \mathbb{E}_{z\sim p(z)}[1 - log\ D_{\theta_d}(G_{\theta_g}(z))]$ and Gradient descent on G: $\underset{\theta_g}{min}\ \mathbb{E}_{z\sim p(z)}[1 - log\ D_{\theta_d}(G_{\theta_g}(z))]$. Gradients are small when performing gradient descent on G at first, so generator learns nothing while the discriminator perfects the task and the equilibrium never achieved. Instead we can perform gradient ascent on G: $\underset{\theta_g}{max}\ \mathbb{E}_{z\sim p(z)}[log\ D_{\theta_d}(G_{\theta_g}(z))]$

Issue 2: Mode collapse: this happens when the generator focuses on just one mode of the data which heavily reduces the sample diversity. Solution? a possible remedy is that run generator for more than one step so that it will be prepare for future discriminator and won't overfit to current state of the discriminator.

GAN Variations

Generate label specific images: this requires the dataset to have the appropriate labels. The training would look exactly the same as in the original algorithm, with the exception of passing the labels. $V(G, D) = \mathbb{E}_{x\sim p_d}[log(D(x|y))] + \mathbb{E}_{z\sim p_z}[log(1 - D(G(x|y)|y))]$
Pix2Pix: given paired (x, y) images for training, learn to transfer x to y(painting and real images as an example). Formulation:
$\mathcal{L}(G, D) = \mathbb{E}_{x,y}[log\ D(x, y)] + \mathbb{E}_{x,z}[1 - log\ D(x, G(x, z))] + \mathbb{E}_{x,y,z}[\|y - G(x, z)\|_1]$
CycleGAN – unpaired image translation:
$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X) + \lambda\mathcal{L}_{cyc}(G, F)$, $\mathcal{L}_{GAN}$ is normal GAN loss and $\mathcal{L}_{cyc}$ is:
$\mathbb{E}_{x\sim p_d(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y\sim p_d(y)}[\|G(F(y)) - y\|_1]$

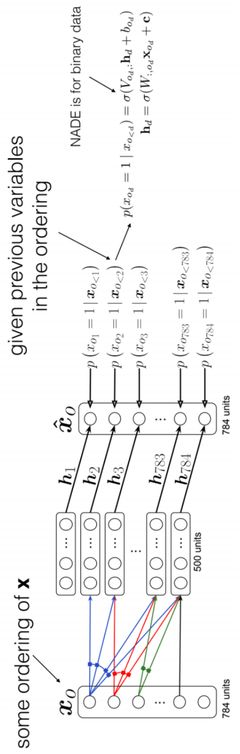
GAN vs. VAE

What are the benefits? Implicit pdf in VAE with intractable data likelihood.No variational bound is needed.Sharper images.Specific model families usable within GANs are already known to be universal approximators, so GANs are already known to be asymptotically consistent. Some VAEs are conjectured to be asymptotically consistent, but this is not yet proven. What are the weaknesses? GANs only care about samples, there is no latent space.

Autoregressive Models

$$\prod_1^n p(x_i|x_{<i})$$

NADE



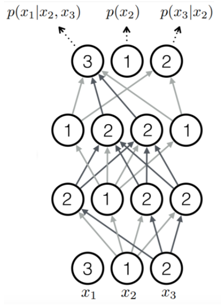
NADE is for binary data
$p(x_{o_d} = 1 \mid x_{o_{<d}}) = \sigma(V_{o_d,:}\mathbf{h}_d + b_{o_d})$
$\mathbf{h}_d = \sigma(W_{:,o_{<d}}\mathbf{x}_{o_{<d}} + \mathbf{c})$
given previous variables in the ordering
$p(x_{o_1} = 1 \mid \boldsymbol{x}_{o_{<1}})$
$p(x_{o_2} = 1 \mid \boldsymbol{x}_{o_{<2}})$
$p(x_{o_3} = 1 \mid \boldsymbol{x}_{o_{<3}})$
$p(x_{o_{783}} = 1 \mid \boldsymbol{x}_{o_{<783}})$
$p(x_{o_{784}} = 1 \mid \boldsymbol{x}_{o_{<784}})$
$\hat{\boldsymbol{x}}_O$
784 units
$\boldsymbol{h}_1$ $\boldsymbol{h}_2$ $\boldsymbol{h}_3$ $\boldsymbol{h}_{783}$ $\boldsymbol{h}_{784}$
500 units
some ordering of $\mathbf{x}$
$\boldsymbol{x}_O$
784 units

Training:

$$\frac{1}{T}\sum_{t=1}^{T} log(p(x^t)) = \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{D} log(p(x_i^t|x_{<i}^t))$$

Advantages: efficient: computations are in O(TD), could make use of second-order optimizers, easily extendable to other types of observations, random order works fine!

MADE



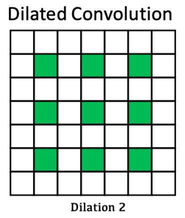
Masked Autoencoder Distribution Estimator: Constrain Autoencoder such that output can be used as conditionals $p(x_i|x_{<i})$ to fulfill autoregressive property. Randomly sample numbers and connect nodes in a layer to nodes in the upper layer only if the upper layer node has a number bigger or equal to the lower layer node number.
pros? Training has the same complexity as regular autoencoders, Computing $p(x)$ is just a matter of performing a forward pass however, Sampling however requires D forward passes.
cons? In practice, very large hidden layers necessary.

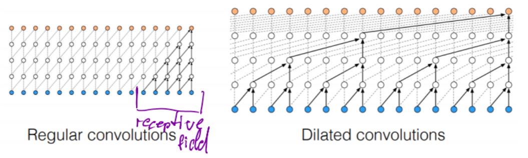
PixelRNN/CNN

Generate image pixels starting from corner. In **PixelRNN** Dependency on previous pixels modeled using an RNN (LSTM). Issue: Sequential generation is slow – due to explicit pixel dependencies. In **PixelCNN** Dependency on previous pixels now modeled using a CNN over context region. PixelCNN – Masking over color channels: $p(x_i|x_{<i}) = p(x_{i,R}|x_{<i})p(x_{i,G}|x_{i,R}, x_{<i})p(x_{i,B}|x_{i,G}, x_{i,R}, x_{<i})$.
Masking Problem? stacking layers of masked convolution creates a blindspot. solution? use two stacks of convolution, a vertical stack and a horizontal stack. Benefits? Training is faster than PixelRNN (can parallelize convolutions since context region values known from training images) Generation must still proceed sequentially $\Rightarrow$ still slow.

TCNs - WaveNet

Adapt PixelCNN to work with Audio data.Problem: much larger dimensionality than images and large-scale temporal dependencies. Solution? use dilated convolution to increase the receptive field.





Autoregressive vs. GAN

pros:explicit likelihood $p(x)$ so training is easier and likelihood of training data gives good evaluation metric. cons: sequential generation is SLOW!

RL

We model the world as an MDP.

- set of states S
- set of actions A
- reward function: $r : S \times A \to \mathbb{R}$
- transition function $p : S \times A \to S$
- initial state S_0
- discount factor γ

$$G_t = \sum_{k=0}^{\infty}\gamma^k R_{t+k+1}$$
$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$
$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s]$$
$$= \sum_a \pi(a|s)\sum_{s'}\sum_r p(s', r|s, a)[r + \gamma v_\pi(s')]$$

How to solve Bellman equation

1. DP:
   - Requires MDP with given environment model (transition probabilities)
   - Value function based on estimates of subsequent values
2. Temporal-Difference Learning
   - No need to known environment model
   - Combination of DP and MC principles
   - Value function updated based on estimates of subsequent values along trajectory

Value Iteration - DP

initialize $V_0(s) = 0$ in iteration k update the values:
$V_k(s) = \underset{a\in A}{max}\sum_{s',r} p(s', r|s, a)[r + \gamma V_{k-1}(s')]$
Pros: Exact methods. Policy/Value iteration are guaranteed to converge in finite number of iterations.Value iteration is typically more efficient. **Cons**: Need to know the transition probability matrix. Need to iterate over the whole state space (very expensive). Requires memory proportional to the size of the state space.

Temporal Difference Learning

Given a policy, for each action a transitioning from s to s', compute the difference to current estimate and update value function:
$\Delta V(s) = r(s,a) + \gamma V(s') - V(s)$,
$V(s) \leftarrow V(s) + \alpha \Delta V(s)$ We don't update the whole state space, only visited states.
Pros: Less variance than Monte Carlo Sampling due to bootstrapping.More sample efficient.Do not need to know the transition probability matrix. **Cons**: Biased due to bootstrapping. Exploration/Exploitation dilemma. Can behave poorly in stochastic environments.

Exploration vs. Exploitation Dilemma

Random policy $\rightarrow$ Exploration $\rightarrow$ Visit states close to starting point more often, Far away states get neglected.
Greedy policy $\rightarrow$ Exploitation $\rightarrow$ Can find reward quickly, Getting stuck in local minimum.

Two Major TD Implementations

- SARSA, **On-Policy**, Computes the Q-Value according to a policy and then the agent follows that policy.
- Q-Learning, **Off-Policy**, Computes the Q-Value according to a greedy policy, but the agent follows a different exploration policy.

Q Learning

Q Function: $Q_\pi(s,a) = \mathbb{E}_\pi[G_{t+1}|S_t = s, A_t = a]$

$$\overbrace{\boxed{\Delta Q(S,A) = R_{t+1}}}^{\text{Data from exploration policy}} + \gamma \overbrace{\boxed{\max_a\{Q(S',a)\}}}^{\text{greedy policy to update Q function}} - Q(S,A)$$

, $Q(S,A) \leftarrow Q(S,A) + \alpha \Delta Q(S,A)$

Deep Q-Learning

Scale Q Learning to also work in non tabular settings.

$$loss(\theta) = \| \overbrace{\boxed{(R + \gamma \max_{a'}\{Q_\theta(S',a')\})}}^{\text{we don't backprop gradients here}} - Q_\theta(S,A)\|^2$$

Replay Buffer

SGD assumes that the updates we apply are i.i.d. In RL, values of states visited in a trajectory are strongly correlated.Use a replay buffer to store generated samples. During updates, randomly sample transitions from the buffer. Since Q-learning is off-policy, we are allowed to use old samples.

Policy Search Methods

Q-learning is limited to discrete action spaces so for continuous action space the problem is intractable. Learning a policy is often much easier. The algorithm directly learns the correct behavior without exploring the value function.

Policy Gradients

rollout trajectories: $p(\tau) = p(s_1, a_1, ..., s_T, a_T) = p(s_1)\prod_{t=1}^T \pi(a_t|s_t)p(s_{t+1}|s_t,a_t)$. Policy is a stochastic function approximator: $\pi(a_t|s_t,\theta): s_t \rightarrow \mu_t, \Sigma_t$. How to **evaluate a policy**? $J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}[\sum_t \gamma^t r(s_t,a_t)]$. **Policy Update**: $\theta = \theta + \nabla_\theta J(\theta)$. How to compute the gradient without knowledge of the environment dynamics:

$$J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}[\sum_t \gamma^t r(s_t,a_t)] = \mathbb{E}_{\tau\sim p_\theta(\tau)}[r(\tau)]$$
$$= \int p(\tau) r(\tau) d\tau \Rightarrow \nabla_\theta J(\theta) = \int \nabla_\theta p(\tau) r(\tau) d\tau$$
$$= \int p(\tau) \nabla_\theta log(p(\tau)) r(\tau) d\tau$$
$$= \mathbb{E}_{\tau\sim p_\theta(\tau)}[\nabla_\theta log(p(\tau)) r(\tau)]$$

To compute $\nabla_\theta log(p(\tau))$:

$$\nabla_\theta log(p(\tau)) = \nabla_\theta log(p(s_1)) + \sum \nabla_\theta log\ \pi_\theta(a_t|s_t)$$
$$+ \sum \nabla_\theta log\ p(s_{t+1}|a_t,s_t)$$
$$= \sum \nabla_\theta log\ \pi_\theta(a_t|s_t)$$

doesn't depend on environment

$$\nabla_\theta J(\theta) = \mathbb{E}_{p(\tau)}[\overbrace{\boxed{(\sum_{t=0}^T \nabla_\theta log\ \pi_\theta(a_t^i|s_t^i))}}^{\text{Maximum Likelihood}}$$
$$\overbrace{\boxed{(\sum_{t=0}^T \gamma^t r(s_t^i,a_t^i))}}^{\text{The trajectory reward scales the gradient.}}]$$

REINFORCE – the variance problem

REINFORCE updates θ by using sampling to achieves gradients. What happens if all cumulative rewards are positive? All trajectories are made more likely. What happens if the biggest trajectory reward is 0? Probability of these trajectories is not increased!
To reduce the variance we can introduce a baseline $b(s_t^i)$

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_{i=1}^N(\sum_{t=0}^T \nabla_\theta log\ \pi_\theta(a_t^i|s_t^i))$$
$$(\sum_{t=0}^T \gamma^t r(s_t^i,a_t^i) - b(s_t^i))$$

Policy Optimization vs. Dynamic Programming

Policy is Directly optimize desired quantity vs. in DP Indirect, exploit problem structure and self-consistency. PO is More compatible with modern ML machinery, including NN and recurrence vs DP is More compatible with off-policy and exploration. PO is More versatile and flexible and More compatible with auxiliary objectives vs DP is More sample efficient (when they work).

Actor-critic

The policy (actor) and the value function (critic) are both represented by neural networks.

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_{i=1}^N\sum_{t=0}^T \nabla_\theta log\ \pi_\theta(a_t^i|s_t^i)$$
$$(\overbrace{\boxed{r(s_t^i,a_t^i) + \gamma V(s_{t+1}^i)}}^{\text{Function Approximation instead of whole trajectory rollouts}} - V(s_t^i))$$

Actor-critic: TRPO

Trust Region Policy Optimization: The idea is to control step length by preventing the action distribution to change too much. It is a second order optimization problem and computationally heavy:

$$max\ \mathbb{E}_t[\frac{\pi(a_t|s_t,\theta)}{\pi(a_t|s_t,\theta_{old})}A_t] \quad A_t := \text{advantage func.}$$
$$\text{s.t. } \mathbb{E}_t[D_{KL}(\pi(.|s_t,\theta_{old}),\pi(.|s_t,\theta))] \leq \delta$$

Actor-critic: PPO

Follow up work of TRPO, but using first order optimization.

$$max\ \mathbb{E}_t[\frac{\pi(a_t|s_t,\theta)}{\pi(a_t|s_t,\theta_{old})}A_t] -$$
$$\beta\, \mathbb{E}_t[D_{KL}(\pi(.|s_t,\theta_{old}),\pi(.|s_t,\theta))]$$

Even simpler, we can even do better by using a clipped objective!

$$L^{CLIP} =$$
$$max\ \mathbb{E}_t[\sum_{t=0}^T[min(r_t(\theta), clip(r_t(\theta), 1-\epsilon, 1+\epsilon))A_t]$$

where $r_t(\theta) = \dfrac{\pi(a_t|s_t,\theta)}{\pi(a_t|s_t,\theta_{old})}$

Actor-critic: DDPG

DDPG is essentially a combination of insights from DQN and Deterministic Policy Gradients (DPG).