Modular Learning

Definition: A family of parametric, non-linear and hierarchical representation learning functions, which are massively optimized with stochastic gradient descent to encode domain knowledge, i.e. domain invariances, stationarity.

- Neural Network is a directed acyclic graph
- Use loss function that matches output distribution to improve numerical stability and make gradients larger
- Input and output distribution of every module should be the same to prevent inconsistent behavior and harder learning

Backprop: chain rule $\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}$, $\nabla_{\boldsymbol{x}} \boldsymbol{z} = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^T \cdot \nabla_{\boldsymbol{y}} \boldsymbol{z}$

1. Compute forward: $a^{(l)} = h^{(l)}\left(x^{(l)}\right)$, $x^{(l+1)} = a^{(l)}$
2. Compute reverse: $\frac{\partial \mathcal{L}}{\partial a^{(l)}} = \left(\frac{\partial a^{(l+1)}}{\partial x^{(l+1)}}\right)^T \cdot \frac{\partial \mathcal{L}}{\partial a^{(l+1)}}$
   $\frac{\partial \mathcal{L}}{\partial \theta^{(l)}} = \frac{\partial a^{(l)}}{\partial x^{(l+1)}} \cdot \left(\frac{\partial \mathcal{L}}{\partial a^{(l)}}\right)^T$
3. Update params: $\theta_{t+1}^{(l)} = \theta_t^{(l)} - \eta \nabla_{\theta_t^{(l)}} \mathcal{L}$

Deep Learning Optimization (1)

Pure optimization very direct goal to optimize (e.g. scheduling). ML wants to optimize test error that is intractable/only indirectly optimizable. Reduce different cost function on training set, optimum might be not optimal for test set (overfitting).

Gradient descent: dataset mostly too large, slow, not better optimum/faster convergence. SGD: standard error $\sigma/\sqrt{m}$, noisy gradients act as regularizer, dynamicly changing data possible.

Ill conditioning: if 2nd order change is greater than 1st ($\frac{1}{2}\epsilon^2 g^T H g > \epsilon g^T g$), loss increases. Later training, reduce lr
Pathological curvatures: ravine region in loss surface, high gradients in suboptimal direction, oscillations, slow convergence

Hessian: requires large batch to be accurate, hard to compute. $w_{t+1} = w_t - H_{\mathcal{L}}^{-1}\eta_t g_t$
Momentum: maintain momentum from previous updates to dampen oscillations: $u_{t+1} = \gamma u_t - \eta_t g_t$, $w_{t+1} = w_t + u_{t+1}$. Exponential averaging $\Rightarrow$ more robust gradients, faster
Nesterov momentum: take future gradients, better in theory.

RMSprop: adaptive lr, exp. averaging over norms, assuming directions of sensitivity axis aligned. $r_t = \alpha \cdot r_{t-1} + (1-\alpha)\cdot g_t^2$, $\eta_t = \frac{\eta}{\sqrt{r_t}+\epsilon}$, $w_{t+1} = w_t - \eta_t \cdot g_t$

AdaGrad: adaptive lr, *sums* norm, thus based on scale and frequency, bad for nonconvex. Very similar to RMSprop, but sums the scales over all time steps (Gt) instead of exponentially averaging. $r_t = r_{t-1} + \text{diag}(g_t^2)$

Adam: Combine adaptive lr and momentum (applied on unscaled gradients). Bias correction to account init at origin.

Bayesian optimization: gradient-free, educated trail and error guesser that works well for low dimensional spaces (up to 1000, but no more than 20-50 parameters usually.) Determine next point on uncertainty and expectation

Deep Learning Optimization (2)

Normalization: center data around 0, scale input variables to get same variance, and de-correlate features (to remove inductive bias). Allows higher learning rate and better learning. *Batch-Norm*: ensure Gaussian distribution of features over batches. $\hat{y}_i = \gamma \cdot \hat{x}_i + \beta$
$\mu_B = \frac{1}{m}\sum_{i=1}^m x_i$, $\sigma_B^2 = \frac{1}{m}\sum_{i=1}^m (x_i - \mu_B)^2$, $\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma^2+\epsilon}}$
Reduce effect of 2nd order between layers, acts as regularizer by introducing noise, let network control mean and variance.
During testing, take moving average of last training steps
Regularization: objective during training to reduce test error
ℓ_2: introduce objective $\frac{\lambda}{2}\sum_l ||w_l||^2$, weight decay for SGD
ℓ_1: sparse weights with $\lambda \sum_l ||w_l||$
Others: Dropout, Early stopping, Augmentation, Multitask

Weight initialization: small weights to keep data at origin, large to have strong gradients, preserve variance of activations ($w \sim \mathcal{N}(0, \sqrt{1/d})$), no learning if all same, prevent dead ReLU

(Modern) Convolutional Neural Networks

Images stationary signals with spatial structure and huge dimensionality. Dimensions highly correlated (translation inv)

Transfer Learning: use large datasets to learn useful features, prevent overfitting, fine-tune less layers if datasets similar, use lower lr for pre-trained layers as close to optimum

(a)If both task have the same labels, we can initialize all layers. Otherwise, the classification layer (last layer) must be newly trained. If there is only very few data available, only fine-tune this layer. (b) If datasets are very different, the fully connected layers need to be replaced.
Architectures: small filter for less params and higher non-linearity (even $n \times 1/1 \times n$), different scales on same input (stack of convs prone to overfitting), vanishing gradients by intermediate classifiers or residual connections (learn difference instead of mapping) $H(x) = x + F(x)$, possibly with gates

Tracking: *Fast R-CNN* based on middle feature map, extract BB (selective search, NN for *Faster R-CNN*). RoI pooling to get fixed-size output. *Siamese*: train on similarity of BB patches.

Recurrent Neural Networks

Backprop thorugh time: gradients of weights on memory W:
$\frac{\partial \mathcal{L}}{\partial W} = \sum_{t=1}^T \sum_{k=1}^t \frac{\partial \mathcal{L}_t}{\partial y_t}\frac{\partial y_t}{\partial c_t}\left(\prod_{i=k+1}^t \frac{\partial c_i}{\partial c_{i-1}}\right)\frac{\partial^+ c_k}{\partial W}$
Formulating RNN as $c_t = W \cdot \sigma(c_{t-1}) + U \cdot x_{t-1}$ leads to:
$\left\|\frac{\partial c_{t+1}}{\partial c_t}\right\| \leq \|W^T\| \cdot \left\|\text{diag}\left(\frac{\partial \sigma(c_t)}{\partial c_t}\right)\right\|$. If norm of non-linearity bounded by γ, and $\|W^T\| < 1/\gamma$, then vanishing gradients. If $\|W^T\| \gg 1/\gamma$ and non-linearity not zero, then exploding gradients. Quick fix for second: clip gradient norm

LSTM: Prevent vanishing gradient by gated skip connections over time. Forget, output, and input+candidate gate

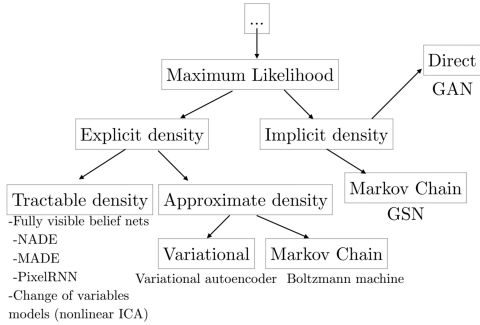
GNN: *Deep Walk*: latent repr. by random walks, skip gram on sequences, not dynamic. *GraphSage*: aggregate information from neighbors, can be mean/max pool with weights, LSTM. *GCN*: $h(H^{(l)}, A) = \sigma\left(D^{-1/2}\hat{A}D^{-1/2}H^{(l)}W^{(l)}\right)$

Deep Generative Models (1)

Generative modeling: learn joint probability $p(x, y)$ or density function $p(x)$. Task can be performed by Bayes: $p(y|x)$. Generalizes better, better modeling of causal relations, out-of-distribution detection $p(y|x)p(x)$ with $p(x)$ low. *Discriminative modeling*: learn pdf $p(y|x)$, task-oriented and mostly better

Applications: RL simulator, creating missing data (pixel patches), super-res., data augm., cross-modal transl. (sketch to img)

Types: *Explicit density*: maximize log likelihood of data by modeling pdf. Must be complex enough and computationally tractable. *Implicit density*: no explicit pdf, only sampling mechanism.



GAN: implicit model, adversarial training. Mini-max game: $\min_G \max_D V(G,D) = \mathbb{E}_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}[\log(D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$. Better loss for generator: $-\log D(G(z))$. Otherwise vanishing gradients if D too strong.

Problems: reaching equilibrium (oscillation around Nash), mode collapse if $\partial\mathcal{L}/\partial z \approx 0$, low dimensional support (JS assumes overlap of distributions).

Improvements: WGAN using Earth-Mover's distance (also good for non-overlapping), usage of labels y like in conditional GANs, label smoothing for overconfident D, Virtual BatchNorm with reference batch to reduce intra-batch inference

Boltzmann machines: Pdf based on energy function we learn: $p(x) = 1/Z\exp(-E(x))$ where $Z = \sum_{x'}\exp(E(x'))$. Z complex, 2^n pos. for binary data. Restrain to pairwise relations: $E(x) = -x^T W x - b^T x$. *Restricted BM*: reduce W by introducing h latents: $E(x,h) = -x^T W h - b^T x - c^T h$, $p(x) = 1/Z\sum_{h'}\exp(-E(x,h'))$, higher-order relations. Can reformulate to $p(h_j|x,\theta) = \sigma(W_{:,j}x + b_j)$, $p(x_i|h,\theta) = \sigma(W_{i,:}h + c_i)$. Maximize log likelihood by contrastive divergence. Sample $h_0 \sim p(h|x)$, $x_1 \sim p(x|h_0)$, a.s.o.

VAE: Model $p(x,z) = p(x|z)p(z)$. Goal is to maximize $p(x) = \int p(x,z)dz$ which is intractable. Use ELBO instead:
$\log p(x) > \mathbb{E}_{q_\varphi(z|x)}[\log p_\theta(x|z)] - \text{KL}(q_\varphi(z|x)||p_\lambda(z))$
Difference is $-\text{KL}(q_\varphi(z|x)||p(z|x))$.

Reparameterization trick: sample from external dist., and transform it to own. For Gaussian: $z = \mu_q + \epsilon \cdot \sigma_q$. Backprop through model params and lower variance than REINFORCE.

Deep Generative Models (2)

Improvements: $q(z|x)$ with NF on top, ELBO is extended by NF term. Optimize prior $p_\lambda(z) = \frac{1}{K}\sum_k q_\varphi(z|u_k)$, u_k trained

NF: Model $p(x)$ directly with series of invertible transformations shifting probability mass. Math expression of NF:
$x = z_k = f_k \circ f_{k-1} \circ ... \circ f_1(z_0) \rightarrow z_i = f_i(z_{i-1})$
$p(z_i) = p(z_{i-1}) \cdot \left|\det \frac{f_i^{-1}}{z_i}\right| \implies p(x) = p(z_0) \cdot \prod_{i=1}^K \left|\det \frac{f_i^{-1}}{z_i}\right|$
$\log p(x) = \log p(z_0) - \sum_{i=1}^K \log \left|\det \frac{f_i}{z_i}\right|$
f must be invertible and has simple det Jacobian (triangular)

Bayesian Deep Learning (1)

Hold dist. per latent variable instead of single val. *Benefits*: ensemble modeling (better acc), uncertainty estimates, prevent overconfidence, model compression (prior towards 0)

Epistemic uncertainty: dataset limits, unseen data, important for safety-critical and small datasets. Posterior $p(w|x,y)$ intractable. *MC dropout*: apply DP during test (Bernoulli-dist over weights). Var approx. uncertainty. Any NN can be made Bayesian with that, but expensive and not accurate. Can also be motivated from Gaussian Processes. Over-param. models better uncert. estm.

Aleatoric uncertainty: data uncertainty due to noise (e.g. bad sensor). *Data-dependent/heteroscedastic*: specific raw inputs hard to interpret, predict uncert. per data point: $\mathcal{L} = \frac{||y_i - \hat{y}_i||^2}{2\sigma_i^2} + \log \sigma_i$. *Task-dependent/homoscedastic*: introduced by task (e.g. depth estimation), Sol: train on multiple tasks. $\mathcal{L} = \frac{||y_i - \hat{y}_i||^2}{2\sigma^2} + \log \sigma$

Bayesian Deep Learning (2)

Bayes by Backprop: approx. true posterior $p(w|\mathcal{D})$ by $q(w|\theta)$: $\mathcal{L} = \log q(w_s|\theta) - \log p(w_s) - \log p(\mathcal{D}|w_s)$ where $w_s \sim q(w_s|\theta)$
Example: assume Gaussian variational posterior with softplus $w = \mu + \epsilon \cdot \log(1 + \exp \rho)$, then learn μ and ρ by SGD.

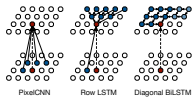
Deep Sequential Models

Autoregressive Models: generative without latent variables, assuming order in data, conditional probs $p(x) = \prod_k p(x_k|x_{<k})$. Not necessarily parameter sharing, $p(x)$ tractable, but slow

NADE: model output with single layer, $\mathcal{O}(D \times H)$ params
$p(x_d = 1|x_{<d}) = \sigma\left(V_{d,:} \cdot h_d + b_d\right)$, $h_d = \sigma\left(W_{:,<d} \cdot x_{<d} + c\right)$
MADE: Autoencoder with carefully masked connections. y_d only depends on $x_{<d}$. Connections can be shared with future d

PixelRNN: row-wise pixel and sequential color generation
$p(x_i|x_{<i}) = p(x_{i,R}|x_{<i}) \cdot p(x_{i,G}|x_{i,R}, x_{<i}) \cdot p(x_{i,B}|x_{i,R}, x_{i,G}, x_{<i})$
Row-LSTM: next output depends on three hidden states above
Diagonal-BiLSTM: use all pixels before (all prev rows and left)
PixelCNN: masked convs to only see top and left. Causes blind spot. Use separated vertical and horizontal stack

PixelVAE: Standard VAE with PixelCNN as decoder



Deep Reinforcement Learning

Value function $q^\pi(s_t, a_t) = \mathbb{E}_\pi\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...|s_t, a_t\right]$
Bellman equation $q^\pi(s_t, a_t) = \mathbb{E}_\pi\left[r_{t+1} + \gamma q^\pi|s_t, a_t\right]$
Optimal policy with $q^*(s_t, a_t) = r_{t+1} + \gamma \max_{a_{t+1}} q(s_{t+1}, a_{t+1})$

Value-based: learn q^* to get π^*. Q-Learning (off-policy):
$\mathcal{L} = \mathbb{E}\left[\left(r + \gamma \max_{a_{t+1}} q(s_{t+1}, a_{t+1}, \theta) - q(s_t, a_t, \theta)\right)^2\right]$
For gradient calculation, bootstrapped val is fixed.
Stability problems: bootstrap, target and policy always changing, oscillations; seq. data break iid assump.; scale of q values hard to control, unstable gradients;
Solutions: experience replay (store samples $\langle s, a, r, s'\rangle$ in dataset, sample from that, makes batch iid), freezing target network every K iters to avoid oscillations, clip rewards, skip frames, control exploration vs. exploitation by annealing ϵ-greedy policy

Policy-based: learn π^* directly, avoid problems with q vals (especially hard for continuous action space). Training steps:

1. Determine q by simulation: $q^{\pi_w}(s_t, a_t) = \mathbb{E}\left[r_t + \gamma r_{t+1} ...|\pi_w\right]$
2. Maximize q by $\frac{\partial \mathcal{L}}{\partial w} = \mathbb{E}\left[\frac{\partial q^\pi(s,a)}{\partial a}\frac{\partial a}{\partial w}\right]$ (deterministic)
   or $\frac{\partial \mathcal{L}}{\partial w} = \mathbb{E}\left[\frac{\partial \log \pi^w(a|s)}{\partial w} q^\pi(s,a)\right]$ (stochastic)

Asynchronous Advantage Actor-Critic: Learn both policy and value function at same time, run multiple agents simultaneously (more diverse samples), advantage estimates: use learned value function to compare actually gained q value. If loss is higher, unexpected (good) things happened ⇒ exploration

Model-based: try to model environment and be aware of rules. E.g. AlphaGo with tree-search guided by CNNs. Two policy networks playing against each other, and a third network to predict $V(s_t) = \sum_{a'} \pi(a'|s_t) \cdot q^\pi(s_t, a')$.

Math to know

Forward KL, $D_{KL}(p||q)$, *overestimates* the variance of the true posterior P by assigning a high probablity mass to our approximate posterior q everywhere that "the data occurs" [everywhere P has some probability mass].
$D_{KL}(p||q) = \int p(x) \log \frac{p(x)}{q(x)} dx \Rightarrow$ if $p(x) > 0$, then $q(x) > 0$

Backward KL, $D_{KL}(q||p)$, *underestimates* the variance of P by assignming low probaility mass to q everywhere "data does not occur" [everywhere P has no probability mass].
$D_{KL}(q||p) = \int q(x) \log \frac{q(x)}{p(x)} dx \Rightarrow$ if $p(x) = 0$, then $q(x) = 0$

$a = Wx + b$, $\frac{\partial a_i}{\partial W_{jk}} = 1(i = j) \cdot x_k$, $\frac{\partial a}{\partial b} = \boldsymbol{I}$, and $\frac{\partial a}{\partial x} = W$

Old Exams

Compare non-linear activation functions

ReLU Neural Network is a directed acyclic graph

sigmoid Use loss function that matches output distribution to improve numerical stability and make gradients larger

tanh Input and output distribution of every module should be the same to prevent inconsistent behavior and harder learning

Additional questions

Differences between generative and discriminative models
1. Generative models are used to estimate the joint probability density function $p(x)$. Discriminative models are used, instead, to model the conditional $p(y|x)$.
2. Generative models are often intractable because in the $p(x) = \int p(x|z)p(z)dz$ the integral is not always possible to analytically compute.
3. Discriminative models tend to yield better accuracies given a task, meaning they are optimized for the particular task, at the cost of potential overfitting.

Advantages/Disadvantages of generative models
GAN: Very good, realistic results, fast to sample from, no need to train on likelihood, very flexible to extension *Drawbacks*: no quantitative evaluation, hard to train (sensitive to hyperparameters, mode collapse, etc.), no real objective in terms of likelihood (and distribution is unknown)
VAE: *Benefits*: Usable for data compression, distribution known (calculate likelihood function), stable training (no mode collapse) *Drawbacks*: only approx. likelihood (ELBO), tends to give blurry instead of realistic images, need flexible enough encoder and prior
NF: *Benefits*: directly optimize $p(x)$, one-to-one mapping between z and x (knows exact embedding of any image in latent space) *Drawbacks*: high number of parameters, complexity restrained by requirement of reversible f

Difference RNN/Autoregressive
RNN: shares weigths over steps, applicable to any sequence length, compresses all previous inputs into single hidden state/memory, not necessarily generative
Autoregressive: does not necessarily share weights, fixed in sequence length, are generative

Figures