Gradient Descent

Tuesday, January 9, 2024 6:38 PM

Idea: train model by slowly reducing the loss function on the weights w, inputs X

Batch Gradient Descent: Given some loss function $L(w, X)$:

$$w_i = w_i - \alpha * \frac{\partial L(w, X)}{\partial w_i}$$

Stochastic Gradient Descent: perform the update by taking the loss only at one data point

$$w_i = w_i - \alpha * \frac{\partial L(w, x^n)}{\partial w_i}$$

Minibatch Gradient Descent: perform gradient descent for a reasonable number of examples

$$w_i = w_i - \alpha \frac{\partial L(w, X^B)}{\partial w_i}$$ for batch B

Regression, Perceptron, Logistic

Tuesday, January 9, 2024 6:26 PM

Linear Regression: Given matrix X containing N data points of d dimensions and targets t

Fit the model:
$y = w \cdot x$

By minimizing the sum squared error:
$$SSE = \frac{1}{2}\sum_{n=1}^{N}\left(y^n - t^n\right)^2 = \frac{1}{2}|Xw - t|^2$$

With the solution: $w = \left(X^T X\right)^{-1} X^T t$

- Using gradient descent: $\frac{\partial \mathrm{L}}{\partial \mathrm{w_i}} = \frac{1}{\mathrm{N}}\sum_{\mathrm{n=1}}^{\mathrm{N}}\left(\mathrm{t^n} - \mathrm{y^n}\right)\left(-\mathrm{x_i^n}\right)$
  - Therefore: $w_i = w_i + \alpha\frac{1}{N}\sum\left(t^n - y^n\right)x_i^n$
  - Delta rule: $w_i = w_i + \alpha\frac{1}{N}\sum\delta^n \mathrm{x_i^n}$ for the delta between target and prediction

Perceptron: Linear regression with binary threshold

Given weights w and threshold θ
$$y = \begin{cases} 1 \; if \; w \cdot x \geq \theta \\ 0 \; otherwise \end{cases}$$
If we defied $w_0 = -\theta, \mathrm{x_0} = 1$ and have weights w'
$$y = \begin{cases} 1 \; if \; w' \cdot x \geq 0 \\ 0 \; otherwise \end{cases}$$

Learning:
$w_i = w_i + \alpha\left(t - y\right)x_i$, for $\alpha = 1$, $w_i = w_i + \delta x_i$

Multiclass: train multiple outputs, then take the largest

Logistic Regression: Gives a probability on the inputs

$\mathrm{g(a)} = \frac{1}{\mathrm{e^{-a}}}$, $a = w \cdot x$, $y = g(a)$

Using MSE in gradient descent:
$$L(w) = MSE = \frac{1}{N}\sum_{n=1}^{N}\left(y - t\right)^2$$
$$w_i = w_i - \alpha\frac{\partial L}{\partial w_i}$$
$$\frac{\partial L}{\partial w_i} = \frac{1}{N}\sum_{n=1}^{N}\delta^n g(a^n)\left(1 - g(a^n)\right)x_i^n$$

But will cause errors because of the g(a) term. The correct loss function is cross entropy.

Softmax Regression: Multiclass logistic regression

Train w_k for each class $1 \ldots K$, then $a_k = w_k x$ and $y_k = \frac{g(a_k)}{\sum_j^K g\left(a_j\right)}$

Backprop

Thursday, January 11, 2024 5:34 PM

We want to perform gradient descent for each layer:

Given objective function J, then for some layer with inputs i, outputs j and weights w_{ij}:

$$\frac{\partial \mathrm{J}}{\partial \mathrm{w_{ij}}} = \frac{\partial \mathrm{J}}{\partial \mathrm{a_j}} \frac{\partial \mathrm{a_j}}{\partial \mathrm{w_{ij}}} = -\delta_\mathrm{j} \mathrm{z_i}$$

For output units: $\delta = t - y$

For hidden units $j \to k$: $\delta_j = g'\left(a_j\right) \sum_k \delta_k w_{j,k}$

Dealing with Overfitting

Thursday, January 18, 2024 7:55 PM

Early stopping: stop training before overfitting, use validation set to detect the best stop point

More data: Get more/manufacture more data

Regularization:

Minimize the complexity of the network through regularization

Minimize: $J = E + \lambda C$ for E error and C complexity function

L2: minimize $|W|^2$, derivative: $2w$, make weights smaller proprotional to its size
L1: minimize $|W|$, derivative: 1 , make weights smaller at a constant rate
Brummelhart: eliminate small weights

Dropout:

During training, disable some fraction of hidden layers
Explore different models, make layers less dependent on each other

Add noise:

Add gaussian random noise (0 mean) to inputs, hidden units, or output

Normalization, Weight Initialization, Momentum

Thursday, January 18, 2024 8:22 PM

Minibatch:
- Saves memory
- Still more accurate than stochastic

Shuffle the inputs
- Ensure a good distribution of labels in the shuffled batch

PCA: Principle component analysis
- Shifts the mean of the input variables to 0
- Decorrelates the inputs
- Discard dimensions with smallest eigenvalues

For neural nets, also:
- Divide by stdev

- Makes input space have 0 mean and 1 standard deviation

Z-score:
- Shifts the mean of the input variables to 0
- Divide by stdev

Change sigmoid:
- Use 1.7159*tanh(0.667x) for sigmoid function

Initializing weights:
- Want to initialize weights to have 0 mean and 1 standard deviation
- $W = N(0, \sqrt{\frac{1}{i}})$ for layer with i inputs (fan in)

Batch Normalization:
- Normalize inputs to 0 mean and 1 standard deviation over each mini batch
- Perform this for inputs to each layer
- Add learnable parameters to "undo" normalization
- Allows model to choose to learn the mean if important, otherwise it is ignored
- Performs normalization across each dimension

Momentum:
- Keep running average of weight changes
- Add running average to delta rule, allows it to accelerate towards the common direction
- $v(t) = \gamma v(t-1) - \alpha \nabla E$
- $\Delta w = v(t)$
- Calculate the momentum then jumps in the direction of momentum

- Nesterov momentum: make the jump then calculate the momentum and correct

Adaptive Learning Rates:
- Adjust learning rates for each weight
- Use global learning rate and multiply by gain for each weight
- $\Delta w = -\alpha g_{ij} \nabla E$
- Increase gain if gradients are consistent, $g = g(t-1) + 0.5$
- Decrease gain if gradients are changing sign, $g = g(t-1) * 0.95$

Rprop:
- Use only the sign of the gradient
- Weight updates are all the same magnitude
- Works well only for full batch

- Divide gradient by magnitude

RMSprop:
- Keep moving average of squared gradient for each weight
- Divide gradient by moving average

Convolutional Networks

Thursday, January 25, 2024 8:28 PM

1. Locality: Nearby pixels correlate with nearby pixels
2. Stationary statistics: Statistics relatively uniform across image
3. Translation invariance: identity of objects seldom depend on its position
4. Compositionality: objects are made of parts

Convolutional Networks: apply convolutional filters across input

Forward: for each convolution kernel
- Window size: how large of a window to sample the input
- Convolution: take small window of the input and feed forward into neural network, get feature map
- Non-Linear: apply non-linear activation function (ie ReLu)
- Pooling: pool the output together to form one output of the kernel, max or sum or average
- Stride: How far to slide the window each iteration
- Get a feature map for each kernel applied over the whole image
- Stack convolution layers before pooling

Example: $\begin{pmatrix} -2 & -2 & -2 \\ 0 & 0 & 0 \\ 2 & 2 & 2 \end{pmatrix}$ detects horizontal edges

Choosing hyperparameters: need to choose # layers, # features, feature parameters
- Use random search and cross validation

Convolutions + maxpool = translation invariance

Backward: Backprop and delta rule as usual

Depth:
- Depth allows features of features to be learned, features become more abstract in deeper layers
- Deeper networks better as long as gradients pass backwards easily
- Reuse pre-trained network to compute features, then use softmax/logistic units to adapt to certain task

Invariance, Design, Visualization, Tips

Tuesday, January 30, 2024 10:21 PM

Invariance:

- Translation invariance built into the architecture
- Scaling invariance learned from the dataset
- No rotation invariance

Design:

- Design one modular structure then add more
- Network in network: instead of maxpooling, create small FCN to learn how to combine features
- Global average pooling: replace output FCN with pool that averages the output features

- 1x1 convolutional layer: can create kernel of size 1x1 with to control size of next layer

- GoogLeNet:
  - Each module combines 1x1, 3x3 maxpooling, 3x3, 5x5 into a single layer
  - Repeat multiple modules, increasing number of features as it progresses

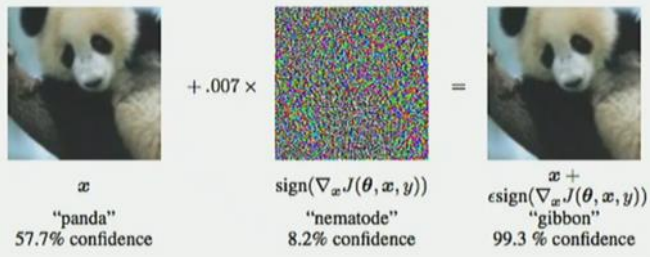
- Residual Networks: introduce pass through into each layer
  - Each layer's output becomes F(x) + x

Visualizing Features: Raw coefficients of filters in higher layers difficult to interpret

- Project activations back into pixel space
  - For maxpool, keep track of which activations were the maximum

- Optimize back to input to maximize a particular feature map or class
  - Take derivative of neuron's activation w/r to input and try to maximize, tells what the neuron will respond to

Adversarial examples: applies to linear classifiers as well



Training tips:

- Start with larger training rate and slow over time
- Ensure feature maps are uncorrelated and have high variance

Objective Functions

Thursday, February 1, 2024 2:01 PM

Maximum likelihood: maximize the probability that the weights explain the data

$$P(W|D) = \frac{P(D|W)P(W)}{P(D)} = \frac{Liklihood * Priori}{Normalizing}$$

Probability of data is normalizing constant
We have no reason (priori) to assume that some W's are better than others
Maximize the modeling of the distribution, P(D|W)

Minimize the negative log likelihood = error: argmin $-\ln\sum(P(x^n))$

In neural netowrks we want to minimize negative loss:

$L = \Pi P(x^n, t^n) = \Pi P(t^n|x^n) * P(x^n)$
$-\ln(L) = -\sum(\ln(P(t^n|x^n)) + \ln(P(x^n)))$

Regression: we assume that the data has some underlying function plus some normal distribution

Minimize the Sum Squared Error: $E = -\frac{1}{2}\sum(t-y)^2$

Logistic Regression & Neural Networks: minimize the difference between output distributions and underlying distributions

Minimize the Cross Entropy Error: $E = -\sum_n \sum_k P(t_k^n) * \ln(P(y_k^n))$

Clustering:

Siamese Neural Networks: performs clustering, dimensionality reduction through supervised learning
- Two identical networks with same weights, linear output
- Trained to minimize distance between two points in same category
- Trained to maximize distance between two points in different categories
- Amplifies dataset, learns on pairs of examples

Loss: $L = (1-Y) * \frac{1}{2}D_w^2 + (Y) * \frac{1}{2}\{\max(0, m - D_w)\}^2$
For D distance between outputs of the networks
M is some margin
Y is 1 for different pairs, 0 for same pairs

In General: the assumption of data distribution leads to different objective functions

- Gaussian -> SSE
- Bernoulli -> cross entropy
- Multinomial -> cross entropy

Recurrent Neural Networks

Thursday, February 1, 2024 2:58 PM

Idea: want to model sequences over time

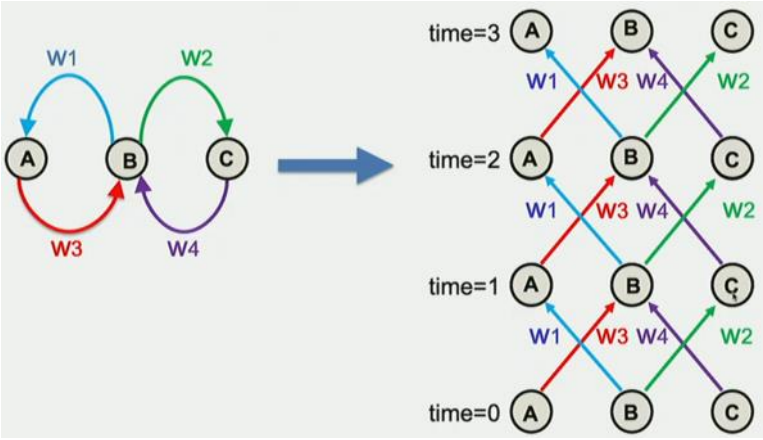
- Map time into space (map time into fixed input window and move window)
  - Autoregressive models: predict next element from some previous elements
- Map time into the state of the network
  - Recurrent networks: cyclical network, stores state based on what came before

RNNs: maps time into state using feedback loops

Jordan networks: Output feeds back into hidden layer
Elman networks: hidden layers feed back into themselves

BPTT(1): backprop deltas calculated for one time delta

Backpropagation:



Propagate activations forward in time $z_A(t), z_B(t), z_C(t)$
Propagate deltas backward
Weight change is the average of weight changes over time periods

RNN on long sequences can lead to exploding or vanishing gradients

Can specify targets for:
- All time layers
- Final time layer
- Subset of time layers

Training: curriculum training

Start with small dataset and increase size as it converges to decent error

LSTM, Uses

Tuesday, February 6, 2024 11:48 PM

Idea: want RNNs to remember things for a long time

LSTM Cell:

- Linear unit with self-link and weight 1
- Information stored when write gate is on
- Information read when read gate is on
- Information kept when keep gate is on

- Resolves the issue of vanishing gradients
- Can latch memory to remember things for longer

Uses:

Use RNNs to generate data
- Train on large corpus of words and predict the next word

Use RNNs to translate sequences

Use convnet + RNN to create image labeling/captioning models

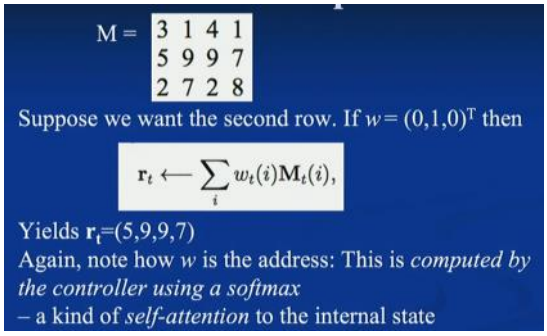
Neural Turing Machine

Thursday, February 8, 2024 8:01 PM

Train recurrent neural network that can interface from structured memory to simulate a turing machine

- Need to make addressing differentiable for both content and location based addressing
- Memory made of matrix of linear neurons, learns to use softmax and weighting to address specific neurons from memory
- Content based addressing: find memory by content

Ex:



If $w = (\frac{1}{2}, \frac{1}{2}, 0)$ then it would return the average of first two rows

Writes performed by gating, wiping memory and then adding to the neurons $\mathrm{m}(t) = a(t) * w(t)$
- Two steps because there are no arithmetic operations for overwriting
- Could add a delta vector, but that would require 3 steps: read current state, calculate delta, add delta to memory

Computing the address:
- k_t: key vector for contrent addressable memory
- β_t: gain parameter on content match
- g_t: switch gate between content and location based addressing
- $\mathrm{s_t}$: shift vector to increment or decrement address
- γ_t: gain parameter on softmax address, making it more binary

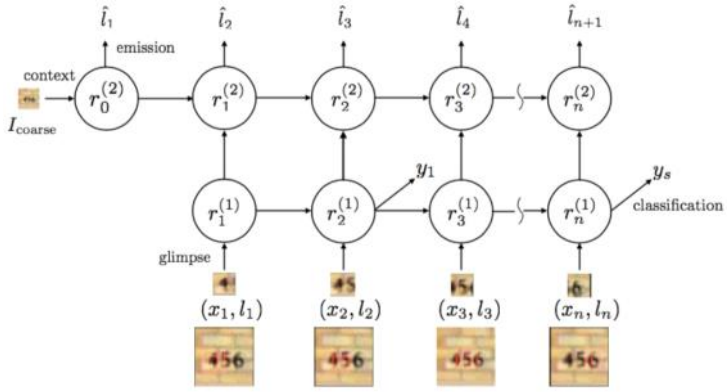
All operations are differentiable w/r to the parameters so we can backprop through them

Can perform copy, repeated copy, associative recall (remember pairs of words), priority sort

RAM Net

Saturday, February 17, 2024 11:01 PM

Recurrent Attention Model: RNN model what can focus on specific parts of images and decide where to look next



Top RNN: controls where to look next
Bottom RNN: performs classification task looking at the region that the controller network output
"Context" & "Glimpse": 3 layer CNN with no pooling
"Emission": 1 hidden layer feed forward network and softmax output

Training:

- Classification: as usual, compare y with target and perform backprop
- Location: reinforcement learning, reward network when it picks a location that works well
  - Start with random locations and encourage it to *explore*, later *exploit* what it has learned

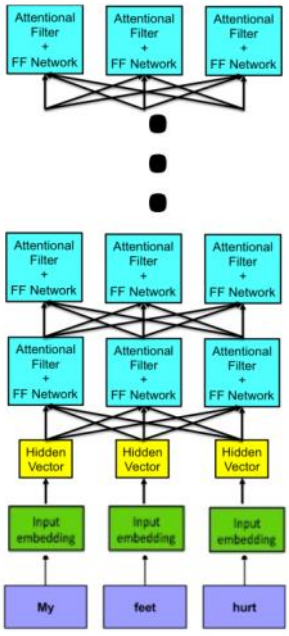
Transformer Networks

Saturday, February 17, 2024 11:25 PM

Idea: want to transform time into space via attention, bypass need to use RNNs or CNNs

Transformer Networks: feedforward networks

Process whole input in parallel without relying on previous states



Each input is put through the same embedding and hidden vector

Encoded inputs can then "communicate" through linked attentions

Each "tower" of attention filters are identical, but are different between each layer
- Each layer computes a different function

Each tower generates a vector of what it's looking for, and what it has
- The inputs to the attention layer are the outputs of the previous layer.
- They come in through three weight matrices
- The Key and Query matrices produce vectors. Their inner product goes into a SoftMax
  - Key: what I have
  - Query: what I'm looking for
- Then the Value matrix output is weighted by the SoftMax.
- This is a form of self-attention

$$K = W_{key}H, \; Q = W_{query}H, \; V = W_{value}H$$
$$O = softmax(KQ) * V$$

Use 8 attention "heads" for each tower, repeated with shared weights for each input dimension

Learns the relationships between what each input represents, what follow up "question" to ask, and what to pass onto the next layer

Much faster training than RNNs, inputs processed in parallel
Allows the model to attend to different parts of the input when making predictions
Encoder focuses on different parts of the input sequence to generate a representation of the input sequence

Image Transformer (ViT):

- Same architecture
- Trained on large dataset

- Cut image into patches, feed into linear projection layer to learn positional embeddings

Universal Transformer Networks:

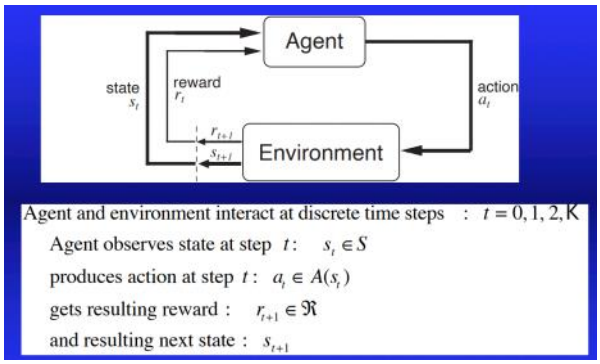
- All attentional filter layer have shared weights
- Effectively an unrolled recurrent neural network
- But, each tower decide adaptively how many iterations to run

- Perform better than Transformer Nets

Reinforcement Learning

Wednesday, February 21, 2024 5:42 PM

Idea: train model on arbitrary task by reinforcement feedback to learn a policy



Policy: models actions probability given state

- Increase probability that a good move is played rather than a bad one

- Gradient:
  1. Sample softmax distribution from output
  2. Treat the sample as the "teacher"
  3. Compute weight changes add them to running average of changes
  4. At the end, multiply the weight changes by the sign of the reward
  5. Change the weights after one cycle

  Over time, good actions will become more likely

  Maximize long term expected reward:
  $R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots$
  For $0 \leq \gamma \leq 1$ (shortsighted vs farsighted)

  Assumes the Markov property (the current action only relies on the current state)

  Value of a state: $V^{\pi}(s)$ expected long term return of being in the state by following the policy

Exploration: want to explore different options rather than picking the best known paths, may discover better ones

  Epsilon greedy: pick a random action with probability ϵ
  - Start with large ϵ and decrease over time

Q-Learning: learning a state-action function

$$q(s, a) = E\left[R_t \middle| S_t = s, A_t = a\right]$$

  Does not require a world model, just learn the next state given the current state
  Use the Q-value of current state to update the previous state, propagate Q-values back to goal state over time

Minimax: Maximize the expected long term reward using a heuristic function

  Initial state: Board position and whose turn it is
  Operators: legal moves that player can make
  Terminal test: test of the state to tell if game is over
  Value function: numeric value for outcome of game

TD-Gammon: neural network function approximator learned to estimate value for each player
- Temporal difference rule: delta is proportional to the difference in next board state and current board state
- $w_{t+1} = w_t + \alpha\left(Y_{t+1} - y_t\right) \sum_k \lambda^{t-k} \nabla_w \mathrm{Y_k}$

AlphaGo: supervised learning on expert positions
- Train multiple networks
- Shallow network performs rollouts to see what good moves are
- Deep network plays itself and trains by policy gradient
- Value network trained to predict winner from states

- Use Monte Carlo tree search

    - Traverse tree to depth limit L
    - Actions selected by maximum value of move plus exploration term
    - Exploration term increases more a move is not tried
  - Node evaluated by value network, rollout of fast policy network

AlphaGo Zero: learned only by self play
  - Used one network for both value and policy
  - Simplified Monte-Carlo tree search for training
    - Each node keeps estimated Q-value
    - Number of times action tried
    - Upper confidence bounds that shrinks as node is explored more
  - ResNet with batch norm

AlphaStar

Thursday, February 22, 2024 8:27 PM

AlphaStar: trained to play starcraft 2

- Real time game with imperfect information (fog of war)

- Initially trained using supervised learning on replays by expert players
- Then reinforcement learning by continually minimizing the KL-distance to supervised network

- League play: play against other agents designed to exploit certain strategies
  - Main agents will eventually play the game
  - Main agents play against everyone in the pool and themselves
  - Main exploiters play against main agents and past agents
  - League exploiters only play against past agents
  - All agents intermittently add frozen copies of themselves

- Actor-critic architecture, multiple models for encoding inputs, and decoding action outputs
  - Inputs encoded using MLP, transformer, ResNet (minimap)
  - Core using Deep LSTM
  - Output embeddings using MLPs
  - Outputs: action type, delay, selected units, target units, target point
  - Value network

  Limit to 22 moves every 5 seconds

  Results: beats 99.85% of human players

Q-Learning

Tuesday, February 27, 2024 8:48 PM

1. Epsilon-greedy to choose action:
   - Choose random action with probability ϵ
   - Otherwise choose the best action
2. Take action and observe the reward r and new state s'
3. Update the Q-value for the action and previous state

Temporal difference method: minimize difference between current state and states reachable from now

Autoencoders

Thursday, February 22, 2024 8:53 PM

Autoencoders: learn to encode an input to lower dimensional representation

- Similar to PCA, spans the principle subspace

- Typically have smaller hidden layer size than input/output
- Multiple hidden layers can lead to non-linear representations (data manifold)

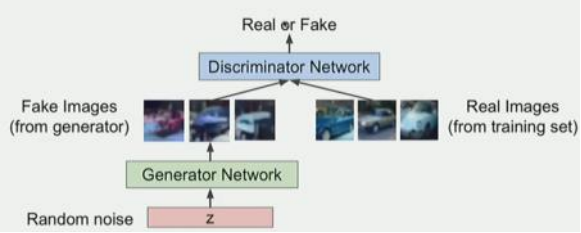
General Adversarial Networks

Thursday, February 22, 2024 9:04 PM

Two networks:

- Generator: decoder network
- Adversary: discriminator network

- Generator tries produce data to fool discriminator into believe it is read data
  - Trained by backprop from going uphill from discriminator
- Discriminator tries to determine if generated image is real or fake data



Similar to games, we can describe the ojective using minimax:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathrm{E}_{x \sim p(data)} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output For real data}} + \mathrm{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output For fake data}}) \right]$$

Training:
- Give generator random noise and generate k outputs
- Take k real samples from training set
- Train discriminator to tell real from fake
- Give generator random noise and generate k outputs
- Train generator objective function holding discriminator fixed

Generator may not train well because discriminator can get ahead of the generator, instead max log(Dd(Gg(z))

Conditional GAN: learn to condition input to output, convert one image to another

For example, replace a horse to zebra in an image

Cycle GAN: Train the model to convert from one image to another and then back as similar as possible

Cycle consistency loss: Minimize the loss between original and cycled image

$$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{x \sim p_{data}(x)}[||F(G(x)) - x||_1] + \mathbb{E}_{y \sim p_{data}(y)}[||G(F(y)) - y||_1]$$

Progressively growing GANs: as models train, grow model size

ReCycle GAN: maps video to video

- Cycle GANs frame-by-frame resulted in loss of temporal information
- Spatial information alone not enough to learn movement, requires temporal knowledge

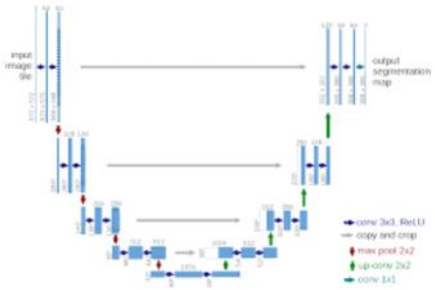
In order to encode *temporal* information, they came up with *Recycle loss*:

$$L_r(G_X, G_Y, P_Y) = \sum ||x_{t+1} - G_X(P_Y(G_Y(x_{1:t})))||^2$$

Here, P_Y is a *predictor network*: it predicts the next frame of video. Here it is trying to predict the next frame in the target domain, then G_X brings it back to the source domain.

So temporal information is encoded by trying to generate the correct next frame in the original domain.

Generator: U-Net architecture using previous 2 frames to predict next frame


input
image
tile
output
segmentation
map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

Contrastive Learning

Tuesday, February 27, 2024 9:29 PM

Metric learning: learning mapping between input and lower dim representation, optimize distances between representations
- Often supervised because we want to cluster inputs

Self-supervision: want to perform supervised learning without labeled data

- Learn features such that $f(I) = f(augment(I))$
- Learns to map an input to an augmented version of the image (cropping, rotation, color mapping)

SimCLR:

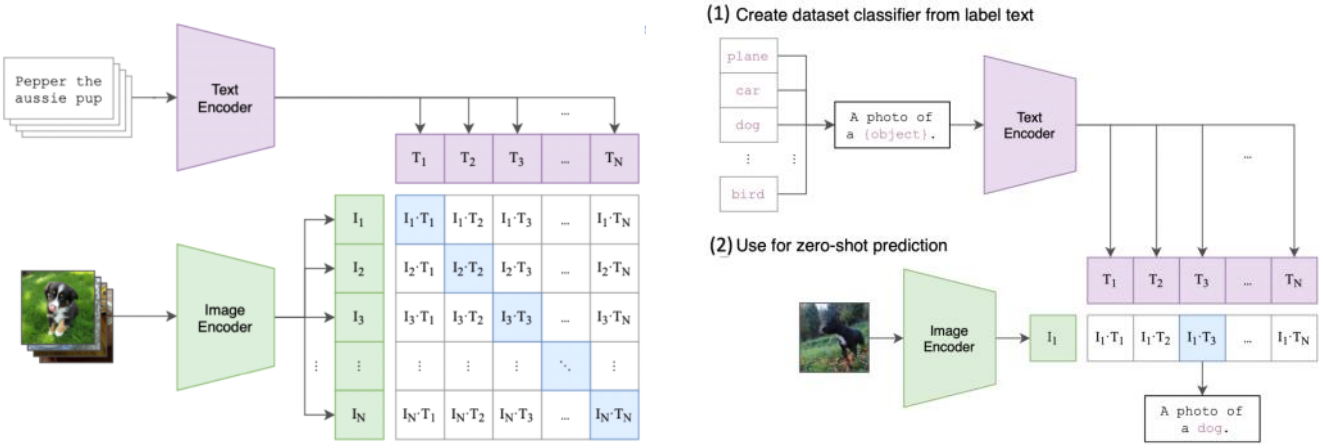
1. Take input and perform 2 transformations and generate representations h using pretrained model (ie ResNet)
2. Feed representations h through 1 hidden layer network with output z and maximize agreement
   - z discards information about color, orientation, etc
   - Maximize agreement between z
   - Discard linear layer
   - Feed h into a non-linear classifier, traing classifier separately
3. Used normalized temperature-scaled cross entropy loss
   - Vectors z are normalized to length 1
   - Use cosine similarity
   - $s_{ij} = \cos(z_i, z_j)$
   - $l(i,j) = -\log\left(\frac{\exp\left(\frac{s_{ij}}{\tau}\right)}{\sum_{k=1\neq i}^{2N} \exp\left(\frac{s_{ij}}{\tau}\right)}\right)$
   - Maximizes cosine distance for correct pair against all incorrect pairs
4. Compose augmentations stochastically (randomly)
   - Apply cropping uniform size
   - Aspect ratio change
   - Apply randomized color change or greyscale
     - Don’t want model to only use color information

DALLE-2, CLIP, Diffusion

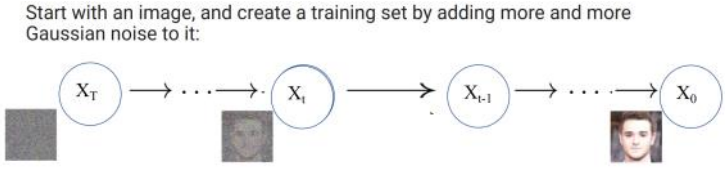
Wednesday, March 6, 2024 12:29 PM

Combines CLIP + Diffusion

CLIP: aligns image internal representations with text representations
- Learn to associate captions and images using contrastive learning
- Transfers without extra training to other tasks achieving good performance
  - Ie ImageNet:
  - Encode labels as "A photo of an X"
  - Encode images and pick the highest match (inner product)
  - Does not do well on extremely niche distributions
- Gathered images using captions which included one of 500k query words
- Use contrastive learning to predict which text as a whole pair with each image
  - Train text encoder to encode captions - transformer
  - Train image encoder to encode images - ResNet
  - Objective: maximize the inner product between associated text and images, minimize unrelated text and images



Diffusion: takes text embeddings and generates images
- Create dataset by adding random Gaussian noise t times
- Train model to take t, random noise, and regenerate the image
- Compute loss and apply gradient descent for each step of denoising



Now use supervised learning to reverse this process!
- U-net architecture
- Time encodings using sinusoidal values (multiple different frequencies can encode large time values)

- Stable Diffusion: model is trained to generate latent (compressed) representations of the images.

Pruning, Lottery Ticket Hypothesis

Wednesday, March 6, 2024 1:54 PM

One-shot pruning:
- Prunes insignificant weights all at once

Iterative magnitude pruning:
- Prune fraction of smallest weights
- Retrain model
- Repeat

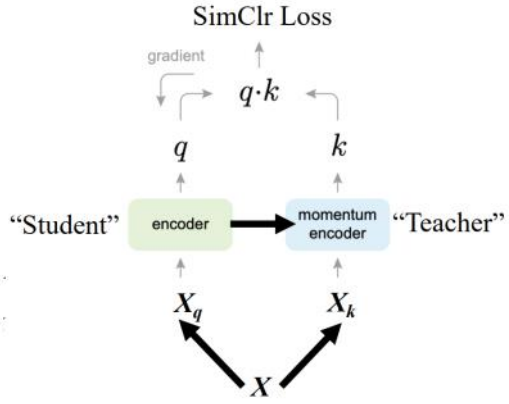
Lottery Ticket Hypothesis:
- A dense network contains a sub network that can reach the same accuracy after the same training time
- Ideal subnet can be identified through pruning
- Late resetting: After pruning, reset weights to partially trained values (ie after 3 epochs)

JEPA, BYOL, Masked Autoencoder

Wednesday, March 6, 2024 3:00 PM

Masked Autoencoder:
- Mask inputs to autoencoder and train to predict masked word
- For images, mask image patches and train model to fill in missing patches

BYOL:
- Train encoder network using a momentum encoder in parallel

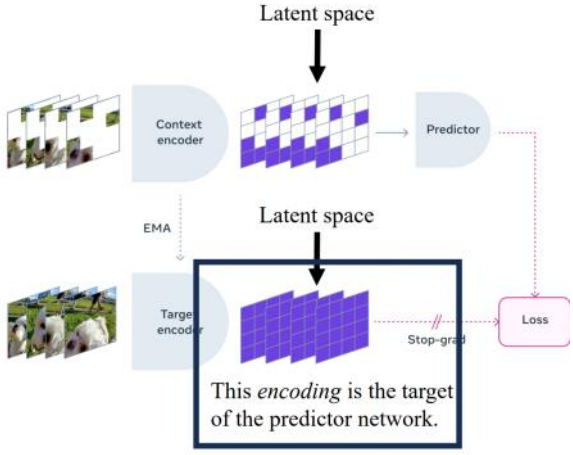


The "teacher" is a network that uses a moving average of the "student" network's parameters.

$\theta_{teacher} = m\theta_{teacher} + (1\text{-m})\theta_{student}$

Where m=0.999

Joint Embedding Predictive Architecture:
- Trains two encoders, one is a exponentially-moving average of the other
- Masks patches of the video and trained to predict the missing patches, masks in space and through time
- Predicts in latent spce
- Encoders are Vision Transformer models



ReZero

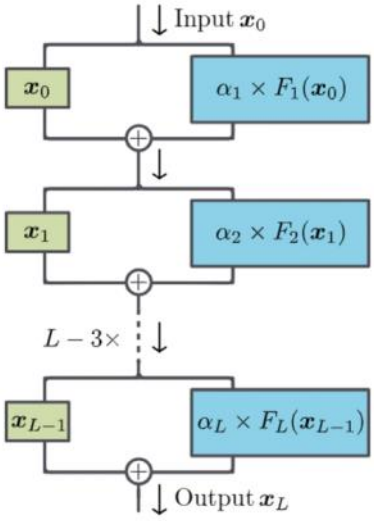
Wednesday, March 6, 2024 3:38 PM

ReZero:

ReZero: **re**sidual with **zero** initialization

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \alpha_i F[\mathcal{W}_i](\boldsymbol{x}_i)$$

Initialize this learned scalar to zero.



- Initializes to identity map, trivially satisfies dynamical isometry
- Train as deep as you want
- Train much faster

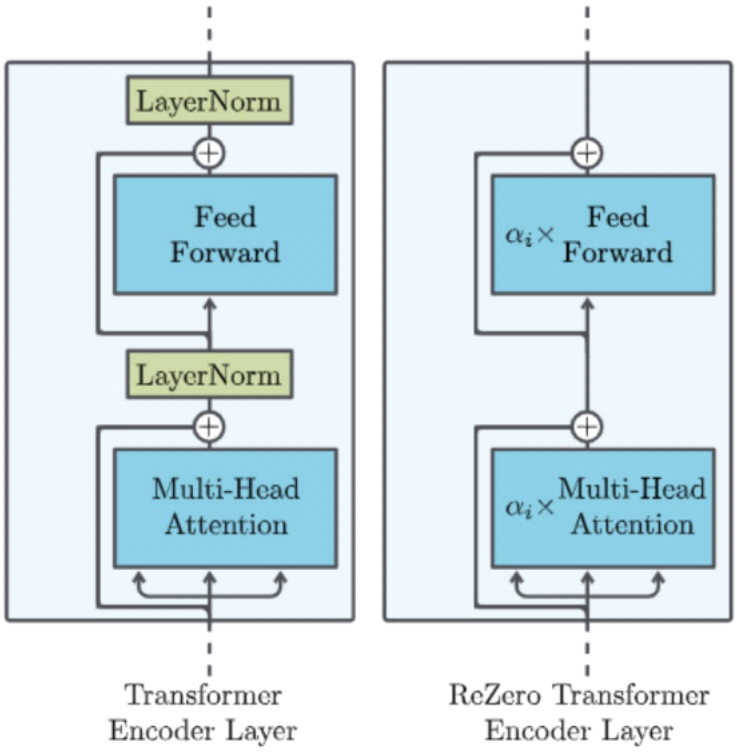
Why would ReZero train faster?

Consider a toy linear residual network that has a single neuron, single weight w and is L layers deep:

$$x_L = (1 + \alpha w)^L x_0$$
$$J_{\text{io}} = (1 + \alpha w)^L$$

When alpha = 1: The network would be very sensitive the small perturbations of the input. You need a learning rate that is exponentially small in depth.
When alpha = 0: The input signal is preserved.

For transformers:


LayerNorm
Feed Forward
LayerNorm
Multi-Head Attention
$\alpha_i \times$ Feed Forward
$\alpha_i \times$ Multi-Head Attention
Transformer Encoder Layer
ReZero Transformer Encoder Layer