LeNet: visual classification network that recognized digits for zipcodes

Representation Learning: learns from input image/output labels

- early layer weights task independent

$\Rightarrow$ customize models trained for diff tasks w less data

Multitask Learning - sharing model weights across tasks improves performance on both

· Deep nets have no obvious perf ceil

$\hat{P}(Y|X) :=$ model (approximation)

Generative: computes full joint $P(X,Y)$

- can generate new data pairs (x_i, y_i)

Discriminative: compute only model target values conditioned on data $P(Y|X)$

generative includes additional assumptions

Generative	Discriminative
· strong assumptions about $P(X,Y)$	- weak assumptions
- insights into phys ps generating data	- little insight into data generation
- faster training	- requires more training data for modest accuracy
+ better performance w sparse data	- fewer forms of bias
- biased if assumptions are violated, poor asymptotic accuracy	- can model more complex datasets
- higher min error	

params in FC layer $= (N \cdot N \cdot W + 1) Y$
$N \cdot N \cdot W$ = shape of input, Y = # neurons

Prediction

- simplify by making strong assumption
  - Y takes a single value given X.

Loss Function measures difference between target prediction and target data value

Linear Regression: $L_2(\hat{y}, y) = (\hat{y} - y)^2$

$$L = \sum_{i=1}^{n} (\hat{Y}_i - Y_i)^2$$

NN have local minima w similar loss values

$$Y_i = a x_i + b$$

Differentiate loss to find optimum values of a, b

- want to minimize expected loss on new data: $E((\hat{y} - y)^2) \stackrel{\sim}{=}$ risk
- actually minimize average loss across a finite number of data points
  - ↳ empirical risk

Cannot do well w empirical risk if
- biased sample
- not enough data

Multivariate: $y = Ax$, $x \in \mathbb{R}^m$, $y \in \mathbb{R}^k$

Gradient := vector of partial derivatives

$\nabla_A L(A) = 0$, all partials are zero, loss not changing $\Rightarrow$ local optimum

Logistic Regression - binary classification

$$f(x) = \frac{1}{1 + \exp(-w^T x)}$$

$\hat{y} = f(x)$ is probability $x \in$ target class

Cross Entropy Loss: negative log probability (softmax) that every label is correct (encourage outputs to 0-1)

$$L = -\sum_{i=1}^{n} Y_i \log \hat{Y}_i + (1 - Y_i) \log(1 - \hat{Y}_i)$$

Compares target distr Y_i w model distr $\hat{Y}_i$

$$L = -\sum_{i=1}^{n} Y_i^T \log \hat{Y}_i$$

Bias - difference between prediction and true y

$$\text{Bias}(\hat{f}(x)) = E[\hat{f}(x) - f(x)] = \bar{f}(x) - f(x)$$

Variance - variance of predictions

Total Squared Error =

$$E((\hat{f}(x) - f(x))^2) = E[(\hat{f}(x) - \bar{f}(x))^2] + \text{Bias}^2 = \text{Variance}(\hat{f}(x)) + \text{Bias}(\hat{f}(x))^2$$

Variance >> Bias^2: too much variation between models. overfitting

Bias^2 >> Variance: models not fitting data well enough. underfitting

Deep Networks: high variance, low bias (complex) ensures $\lambda > 0$

Regularization reduces variance (overfitting)

Positive definite:

M: $v^T M v > 0$, $\forall v \in \mathbb{R}^n$

PSD:

M: $v^T M v \geq 0$ $\forall v \in \mathbb{R}^n$

- all $\lambda_i \geq 0$

Regularized Multivariate Regression:

$$\text{Loss}(A) : \sum_{i=1}^{n} (X_i^T A^T - Y_i^T)(A X_i - Y_i) + \lambda \sum_{i,j} A_{ij}^2$$

- lower variance, higher bias

New formula: $A = M_{yx}(M_{xx} + \lambda I)^{-1}$

Strong regularization (large λ)

$\Rightarrow$ lower variance, higher bias (generalizing model)

Weak Regularization (small λ)

$\rightarrow$ higher variance, lower bias (classify as many correctly as possible)

SVMs: large margins to give room for error

- want to maximize classifiers margin

$f(x) = w^T x + b = 0$ ← boundary

$f(x) = 1$, $f(x) = -1$ (unit length margin)

Constraint: $y f(x) \geq 1$

Hinge Loss:

↳ $\max(0, 1 - y f(x))$

measures how much constraint is violated

↳ as $\|w\|$ increases, hinge loss ↓

To Fix: soft margin SVM

$$\ell = \sum_{i=1}^{n} \max(0, 1 - y_i f(x_i)) + \lambda \|w\|^2$$

- neutralizes effect of increasing $\|w\|$

Nesterov: evaluate gradient at lookahead pos.

MultiClass:

- one vs rest
  - k fn for k classes
- one vs one
  - $\binom{k}{2}$ fn, compare each class against all other each
  - tally votes from all classifiers

$P_i(x)$ = probability $x \in$ class i

$$\text{Loss} = \sum_{j \neq y} \max(0, 1 - (f_y(x) - f_j(x)))$$

Softmax:

$$f_j(z) = \frac{\exp(S_j)}{\exp(S_1) + \ldots + \exp(S_k)}$$

k fold cross validation

- average hyperparams on validation set

partition into K sets, use different set for testing

To reach loss minimum:

- follow negative gradient $-\nabla_W L(W)$

$$W^{t+1} = W^t - \alpha \nabla_W L(W)$$

GD: calculating gradient requires full pass thru dataset - expensive

SGD: minibatches of size m

N/m updates on full pass

· gradient of function orthogonal to contour

Newton's method: compute vector straight to center

↳ quadratic (fast) convergence

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}$$

Update:

$$x_{t+1} = x_t - H_f(x_t)^{-1} \nabla f(x_t)$$

- taking inverse is expensive
- quickly goes to nearest gradient zero - often saddle point

SGD w momentum

$$p^{(t+1)} = \mu p^{(t)} - \alpha g^{(t)}$$

$$w^{t+1} = w^{(t)} + p^{(t+1)}$$

p: momentum

μ: momentum constant

α: learning rate

prevents oscillation, may overshoot

Nesterov: 1) step in gradient dir 2) correct it according to new place

$$p^{(t+1)} = \mu p^{(t)} - \alpha g^{(t)} + \mu p^{(t)}$$

RMSProp scales gradients by inverse of moving average

$$s^{(t)} = \beta s^{(t-1)} + (1 - \beta)(g^{(t)})^2$$

s: Mean squared gradient at time t

β: $[0,1]$ moving average decay factor

$$w^{t+1} = w^t - \alpha \frac{g^{(t)}}{\sqrt{s^{(t)}}}$$

ADAGRAD

- cumulative sum of squared gradients

$$c^{(t)} = \sum_{j=1}^{t} (g^{(j)})^2$$

tends to grow linearly over time, so effective learning rate ↓ $1/\sqrt{t}$

- work well on sets w wide range of gradient magnitudes
- less effective w strong feature dependencies

ADAM: momentum + RMSProp

- compute moving averages of gradient + squared gradient

$$p^t = (\beta_1) p^{t-1} + (1 - \beta_1) g^{(t)}$$

$$s^{(t)} = \beta_2 s^{(t-1)} + (1 - \beta_2)(g^{(t)})^2$$

$$w^{t+1} = w^t - \alpha \frac{p^t}{\sqrt{s^t}}$$

Bias bc moments initialized to 0

Network gets more complex, more local minima near global minima

Evaluate backprop from output back, bc output Jacobian is a row vector

also, common sub expr

Gradient: what happens to loss w weight increase

Grid Search - hyperparams projected on each other

UNet - FC CNN w residuals

Multidimensional arrays incorrectly called tensors

CS 182 MT1

2 Types of Jacobians:

- loss wrt input vector: data path
- loss wrt model params: model path

Convolution: effect of 1 signal on other

$$(h * f)(x, y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} h(i,j) f(x - i, y - j)$$

Correlation: similarity

$$(h \star f)(x, y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} h(i,j) f(x + i, y + j)$$

Num Params:

$n \times m$ filter

ℓ input feature map (depth)

k output feature map - num filters

Conv: $(n \cdot m \cdot \ell + 1) k$

FCC: n input, m output: $(n+1) m$

Convolutional Filters:

$C_{out} \times C_{in} \times F_H \times F_W$

(# output channels, # input channels, filter height, filter width)

· assume nearby pixels share similar structure

· params for filter are shared across image

Input $\rightarrow$ Low level feature $\rightarrow$ high level feature $\rightarrow$ Trainable classifier

Conv Output Size: $(N - F + 2P) / \text{stride} + 1$

Image: $N \times N$, Filter: $F \times F$

Conv Layer:

K: # filters, S: stride, F: filter dim, P: padding

Input of $W_1 \cdot H_1 \cdot D_1$

Produces $W_2 \cdot H_2 \cdot D_2$,

$$W_2 = (W_1 - F + 2P) / S + 1$$

$$H_2 = (H_1 - F + 2P) / S + 1$$

$$D_2 = K$$

$F \cdot F \cdot D_1$ weights per filter

Pooling

- allow flexibility in location of features
- allow union of features to be computed
- reduce output image size

Pooling downsamples. (no params)

LeNet-5 for handwritten digit classification

· 5x5 conv filters at stride 1

[conv-pool] x 2 - conv - fc   1998

AlexNet: similar architecture, bigger/deeper

1st layer: 96 11x11 filters w stride 4

output: [55x55x96]

In: [227x227x3]

~35K params

[conv-pool-norm] x 2 - [conv] x 3 - pool - [fc] x 3

- first use of ReLUs, GPUs, dropout

~60 million params

Transfer Learning

1) Train on ImageNet
2) - small dataset
     · retrain only classifier, i.e. last (end) softmax layer
   - medium dataset
     · perform fine tuning
       - used old weights as initialization, train full network or only some of higher layers
       - maybe retrain lower layers

VGGNet 2014

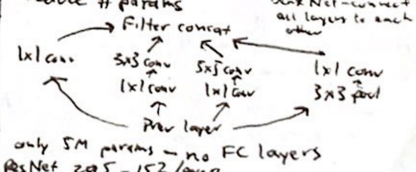
[conv-conv-pool] x 2 - [conv-conv-conv-pool] x 3 - FC x 3   ~138M params

benefits: increased number of layers using only conv ops stacked on top of one another.

GoogLeNet 2014

introduced inception architecture to reduce # params

DenseNet - connect all layers to each other



only 5M params - no FC layers

ResNet 2015 - 152 layers

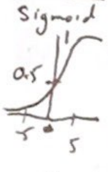
- spatial dim 56x56

use skip conn to propagate faster thru nw

- fewer params than VGG
- batch norm after every conv layer
- no dropout

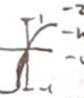
trend toward no POOL/FC layers
- smaller filters, deeper architectures

Sigmoid $\sigma(x) = 1/(1+e^{-x})$



- squashes #s to [0,1]
  - can kill grads
- key in LSTMs
- good for logical fns, learning non linear control
- bad for image nw (ReLU)
- not zero centered

tanh: numbers to [-1, 1]



- zero centered
- kills gradients when saturated
- used in LSTMs for bounded, signed values
- not as good for binary fn

ReLU
$f(x) = \max(0, x)$
- does not saturate in + region
- converges faster than sigmoid/tanh
- not suitable for logical fn
- not for control in RNN
- not zero centered

Leaky ReLU: $f(x) = \max(0.01x, x)$
- will not die

Parametric ReLU: $f(x) = \max(\alpha x, x)$

Exponential Linear Units
$$f(x) = \begin{cases} x & x > 0 \\ \alpha(\exp(x) - 1) & x \le 0 \end{cases}$$
- doesn't die - closer to zero mean outputs

Maxout Neuron
- nonlinearity
- Linear regime, doesn't saturate, doesn't die
- doubles # parameters/neuron

Sigmoids good for smooth fn (robot control), and logical fn (and/or)

If initial weights are too small/large, activations will vanish/explode during fwd pass
Small: variance ↓: no non-linearity
big: variance ↑: activations saturated, gradients → 0
Xavier Initialization keeps variance the same
- pick weights $N \sim (0, \frac{1}{n})$, n = # input neurons

Batch Norm
$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}$$
- improves gradient flow thru nw
- allows higher learning rates
- reduces strong dependence on initialization
- reduces need for dropout
- limits magnitude of gradient

reduces covariance shift

Dropout
- randomly set some neurons to zero in forward pass
1) forces nw to have redundant representation
2) training large ensemble of models that share params
- no dropout during test time
- must scale activations so that for each neuron output at test time = expected output at training time

Inverted Dropout
- divide dropout mask by p at training time

Ensemble Learning
- Bagging (Bootstrap Aggregation)
  - train base models on bootstrap samples
  - take majority vote for classification, average output for regression
  - models trained independently
    - reduces variance in prediction

Boosting
- ordered learners
- each tries to reduce error on examples misclassified by earlier learners
- models are dependent, trained sequentially

ADABOOST: weigh hard samples more

Gradient Boost: use residual to train later models
- reduces bias, possibly variance

Bagging often used w deep learning models, boosting rarely used
→ parallelizes ↳ models don't have much bias

True Ensemble: independent models
- prediction avg: avg prediction probs, or vote
  - always works
- param avg: avg params, almost never works

Model Snapshot: train one model, take snapshots of params
- param avg often works: snapshots close in parameter space

Gradient noise seems to help
- use validation set for hyperparam tuning within each training block
coarse → fine
- want ratio of weight updates / weight magnitudes to be ~ 0.001

Classification
Classification + Localization } single object
Object Detection
Instance Segmentation } multiple

Semantic Segmentation - label every pixel
1. Classify every pixel
   - extract patch, CNN on middle pixel
   - repeat same computation multiple times
2. CNN
   - a bunch of layers to predict all pixels at once
   - convolutions can be expensive
3. FC CNN
   - downsampling/upsampling inside nw

Downsampling: pooling, strided conv
Upsampling:
Nearest Neighbor

$$\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix} \rightarrow \begin{matrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 3 & 3 & 4 & 4 \\ 3 & 3 & 4 & 4 \end{matrix}$$

Bed of Nails

$$\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix} \rightarrow \begin{matrix} 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \\ 3 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}$$

Max Unpooling
- remember which element was max
- Corresponding pairs of up/down sampling layers

Transpose Convolution

4. UNet: FCNN + Residuals
   - residual connection made by copying input + concat w upsampled layer

Classification + Localization

Image → Conv pool → Conv feature → L2 loss → bounding box regression
Conv feature ↘ softmax loss → class scores → classification
Can also share FC layers

Per-Class Regression
- 1 bounding box for each class, choose bounding box by predicted class label

Class Agnostic
- 1 bounding box total

Object Detection
- use metric called "mean average precision = mAP"
- mAP is # from [0, 100], high is good

Detection w Regression
- depending on image, need variable sized outputs

Detection w classification
- need to test many positions and scales
→ only look at promising regions of image

Classification + Region Proposals
R-CNN
1) Input Image
2) Extract region proposals
3. Compute CNN features
4. Classify regions

Problems!
1) finding Region proposals can be hard/time consuming
2) Classifying each part of image is time/space consuming

Fast R-CNN - 25x speedup
- share computation of conv layers between proposals
- put whole image thru ConvNet before extracting regions
- region proposal after conv features, pool

Faster R-CNN - 250x speedup
- after CNN, include a Region Proposal NW
- no need for external region proposals

RPN: - slides small window on feature map
- classify object or not object
- use N anchor boxes at each location
- regression gives offset from anchor boxes
- classification gives prob that each regressed anchor shows object

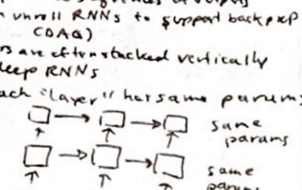
State of the Art: Single Shot Detection
- base boxes centered at each grid cell
Within each:
- regress from each of B base boxes to final box w 5 numbers (dx, dy, dh, dw, confidence)

Output: $7 \times 7 \times (5 \cdot B + C)$
B base boxes, C classes

RetinaNet:
1) fwd pass of ResNet/Conv model
2) each level of downsampling, single shot detection

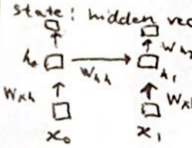
Recurrent NNs
- introduce cycles and a notion of time
- designed to process sequences of data + produce sequences of outputs
- can unroll RNNs to support backprop (DAG)

Layers are often stacked vertically
- deep RNNs
- each "layer" has same params



Lots of flexibility:
1-1: vanilla NN
1-many: image captioning
many-1: sentiment classification
many-many: translation
↳ video classification per frame

At each step: $h_t = f_W(h_{t-1}, x_t)$
same fn, params used at each step
State: hidden vector h



$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$
$y_t = W_{hy} h_t$

Sequence Generation:
- top k sequences generated so far are remembered for next sequence of RNN

Each word of sentence comes from different part of image

W_{hh} is multiplied by gradient at each time step
largest λ of $W_{hh} > 1$, gradients grow exponentially
$\lambda_{max} < 1$, gradients shrink exponentially

LSTMs
- non-linear, linearly transformed hidden states as well as memory cell c_i that isn't transformed
- LSTM encapsulates RNN

RNN:
$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

LSTM:
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

de conv:
$R_i^l = (R_i^{l+1} > 0) \cdot R_i^{l+1}$
guided backprop
$R_i^l = (f_i^l > 0) \cdot (R_i^{l+1} > 0) \cdot R_i^{l+1}$

- remember exactly what came in.
$c_t^l = f \cdot c_{t-1}^l + i \cdot g$ ← c output
$h_t^l = o \cdot \tanh(c_t^l)$ ← h output

"gate" - how much info to let thru
- C_t is filtered version of C_{t-1}
- h_t is output: $\tanh(C_t) \times \ldots$

LSTM:
1) decide what to forget
2) decide what new things to remember
3) decide what to output
- cell c, gradient grow linearly w time
h-path not well behaved

tSNE visualization $O(N^2)$
"Stochastic Neighbor Embedding"
- locally, pairwise distances are conserved
★ similar things end up in similar place
- doesn't cluster data

DBSCAN - density based clustering

What do convnets learn?
- visualize patches that activate neurons
- visualize weights
- visualize representation space
- occlusion experiments
- deconv approaches
- optimization over image approaches

Occlusion - omit part of image to see what classification most heavily depends on

We can generate an image that maximizes some class score
1) feed zeros
2) set gradient of scores vector to be [0, 0, 1, ..., 0], backprop ← 1 in class we're interested in
3) image update
4) fwd image thru nw
5) go to 2.

3 Regularizers:
1) penalize high freq 2) Clip pixels w small norm
3) Clip pixels w small contribution

Saliency Map - image that shows each pixel's unique quality

Deconv - maps features to pixels
Excitatory Input: positive influence on gradient
Inhibitory: negative influence
Linear Nature is primary cause of NN vulnerability to adversarial perturbation
Deep Dream modifies image to boost all activations
Guided Backprop gives good results
↳ requires image regularization - pressure to look like normal images

backprop: $R_i^l = (f_i^l > 0) \cdot R_i^{l+1}$

f: by fwd pass
R: after fwd pass
Abstract Grad.

Semantics - concerned w meaning of texts

Propositional / Formal semantics - block of text converted into formula in logical language

Vector repr - texts embedded into high-dimensional space

Prop: dog bites man → bites(dog, man)

Vec: dog bites man → (0.2, -0.3, 1.5...)

Prop: allows for logical inferences

Vector: word embeddings, bag of words

Bag of Words: sentence is sum of vectors of words in that sentence → no idea of order

Embeddings depend on word similarity → contexts

- words that share similar contexts have similar embeddings

One Hot encoding: vec size = vocab size

Vectors of contexts: counts of nearby words

↳ · vec size = vocab size
 · better captures semantics - similar words have large inner products

Dimension Reduction

BoW Problems
1. treats words as equally similar/dissimilar
2. ignores word ordering

Latent Semantic Analysis:

- dimension of a few hundred
- encodes set of docs in matrix
- T_{ij} is count of word j in document i.
- most entries are 0.

N docs $\boxed{T}$ allows for efficient sparse matrix methods

M word features

Compute $T \approx U^T \cdot V$

$T \approx N \boxed{U^T} \; \boxed{V} \}K$

K latent dims, M features

If $U \perp V$, S is a diagonal matrix of singular values, if t is a document (row of T):

$v = Vt$ is embedding of document in latent space

$t' = V^T V t$ is decoding of document from its embedding $T \simeq U^T S V$

SVD gives best possible reconstructions of documents t' from their embeddings. $t' = V^T V t$

LSA: autoencoding method

LSA uses squared L2 loss for deep net

Word2vec - 2 layer NN: negative sampling

- uses words a few positions away from each center word - "skip grams"
- considers all words as center words, and all context words
- can predict center word from context word, vice versa - (skip-gram) - CBOW
- minimizes softmax loss $-\log p(j|i)$ for each output word given an input word

$$p(j|i) = \frac{\exp(u_j^T v_i)}{\sum_{k=1}^{V} \exp(u_k^T v_i)}$$

j = output word, i = input word, u: output embedding vector, v: input embedding vector

- loss is largest for most similar word (j to i) → don't want similar words consecutively
- deep net actually uses cross entropy - log of softmax

Input x: one hot encoded input word

y: index of output word | implements word analogies

capture more info about relations and properties than LSA · encodes context words in N-dim hidden layer

Criticisms:

- cross entropy loss puts much emphasis on small (unlikely) combinations of word/context
- expensive to normalize softmax ~~term~~ over all words
- uses heuristic down weighting of frequent words

Co-occurrence Matrix

C_{ij} counts # windows containing both words i, j

GloVe - NN used for dimensionality reduction (word analogies)

C_{ij} = # times word j occurs in context of word i.

minimizes: $J(\theta) = \sum_{i,j=1}^{V} f(C_{ij})(u_i^T v_j + b_i + \tilde{b}_j - \log C_{ij})^2$

U_i = word embedding, $\tilde{v}_j$ is context word embedding,

b_i, $\tilde{b}_j$ are bias vectors, $f(.)$ satisfies

$f(0) = 0$, $f(x)$ non decreasing, $f(x)$ saturates

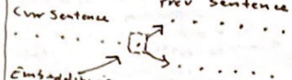
- LSTMs

Lexical Semantics: focuses on meaning of individual words

Compositional Semantics: meaning depends on words, and on how they're combined

Skip Thought Embeddings: use sequence-to-sequence RNNs to next + previous sentences - sentence embeddings

Embedding is ~~[illegible]~~ output state vector of boundary layer



Encoding doesn't require backprop, allows us to encode sentences + paragraphs

Siamese Network for Semantic Relatedness

- trained on pairs of sentences a, b w similarity label y

SICK semantic entailment task:

- score sentences for relations: ENTAILMENT, CONTRADICTION, NEUTRAL

Hard Attention: attend to single input location

- no GD - need RL

Soft Attention:

- compute weighted combo over some inputs using attention nw
- can use backprop to train e2e

Supervised Learning:

- independent input samples, each sample x receives a label y
- pair (x, y) assigned a loss that is differentiable

RL

- learner visits sequence of states in an epoch
- at time t, learner performs action a_t and receives reward r_t from environment
- agent maximizes sum of rewards over 1 epoch

Attention for Recognition: RNN based model

Ba: input entire image — Hard attention

Mnih: glimpses are retinal images - part of image

- glimpse trace on some digit images

Soft Attention for Translation

RNN for Captioning:

- take layer right before classification layer in CNN (features), pass into RNN, start generating text
- parts of description guided by certain parts of image.
- take features before spatial (FC) layer
- pass spatially ordered feature into RNN
  - generate weights per features $L \times D$
- each time, generate attention / data vector
- attention mask a feature, go into next iteration

Soft Attention: generate continuous set of weights, dim = feature dim

Hard: generate set of probabilities that you attend to specific feature vector

Attention weights have same spatial dimensions as last CNN spatial layer

Saliency: product of upstream gradient + feature layer activations → "importance"

Attention maps are explanations of net behavior - identify influential parts of input stream

DRAW: attend to arbitrary regions of input, then start narrowing down to classify

To generate: looks at partial strokes it's drawn itself

- 3 params: x, y, width of box

Models improve accuracy, reduce computation

- learn to predict salience

Translation

Sequence to Sequence RNNs

- Keep n-best list of partial sentences, along with their partial softmax scores

Bleu Scores - compare machine translations against human generated translations, allowing for variation

Unigram Precision: $\frac{\text{correct unigrams in reference sentence}}{\text{unigrams in test sentence}}$

Modified Unigram Precision: clip counts by max occurrence in any reference sentence

Attention heads find start of sentence, subject/object relations, pronouns

N-gram precision: $\frac{\text{correct n-grams in ref sentence}}{\text{n-grams in test sentence}}$

Unigrams capture adequacy

N-grams capture fluency

BLEU score: take weighted geometric mean of n-gram precisions up to some length (~4)

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{N} W_n \log P_n\right)$$

$$BP = \begin{cases} 1, & c > r \\ e^{(1-r/c)}, & c \le r \end{cases}$$

CycleGAN
· 2G, 2D
· 1 V domain
· relies only on real images on other domain for realism
· cycle consistency loss

Sequence 2 Sequence

Encoder (Input) — Decoder (Output)

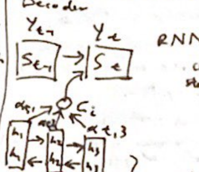
- reversed LSTMs perform the best → first part of sentence is most important, reversal eases long term dependencies from output to input sentence

Beam Search Width := how many phrases are kept track of

Criticism: bottleneck at last unit of encoder

Soft Attention for Translation - Bahdanau 2015

- for each output word, focus attention on a subset of all input words
  - alignment score: how well do input words near j match output words at position i
  - mixture weights: softmax over alignment scores
  - context vector (input to decoder)



compare latent states from encoder/decoder

Criticism: attention fn is complex, but seems to be heat map on word similarity

- data path is complicated, attention path is another recurrent nw between output states

Luong, Pham, Manning 2015 translation

- stacked LSTM w arbitrary depth
- global attention model sits above encoder/decoder, not recurrent
- simpler than Bahdanau's
- LOCAL attention model
  - computes best aligned position first
  - compute context vector centered at that position · simplify + add more recurrence

Parsing

- Sequence models generate linear structures, but these can easily encode trees by "closing parens"

Seq2Seq Parser

1) First tried training a basic seq2seq model on human annotated training treebanks. Poor results (90.5 F1)
2) training on parse trees by Berk parser
3) added attention model (90.5 F1)
4) created synthetic dataset of high confidence parse trees. SotA
   - depth = 3, layer dim = 256
   - dropout between 1+2, 2+3
   - no part of speech tags!
   - input reversing

Problems w Recurrent NW:

- sequential training + inference, hard to parallelize
- long range dependencies have to be remembered across many single time steps
- tricky to learn hierarchical structures

Self Attention

- info flows from within same subnw
- constant path length, supports hierarchical info flow, trivial to parallelize, weighting controls info propagation
- can entirely replace word-based recurrence

Transformer

- stacked with N=6
- positional encoding: words are combined thru attention, location is lost

Attention

Encoder decoder: queries from previous decoder layer, memory keys + values come from output of encoder

Self Attention Layer: all K, V, Q from output of previous layer in encoder

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Multi-Headed Attention allows for per input transformation - attend to multiple parts of input sequence

Position Encoding: spatial encoding

- encoding vector has same dimension as model
- components are all sinusoidal fns of position | Q ~ Luong h_e, K ~ Luong h_s

Large Scale Summarization

- like translation, but no encoder

Memory Networks support

- reading + comprehension
- dialog
- learning from dialog

Cross Attention:
Q: transformed output word embedding
K, V: transformed input word embedding

Reading w Attention over Memory

Framework:

I: Input Feature Map: convert input data to internal feature representation

G: Generalization: update memories given new input

O: produce new output (in feature representation space) given the memories

R: (response) convert output O into a response seen by outside world

Q&A Memory NW — standard key — value

Input data: $m_i = Ax_i$, $c_i = Cx_i$ → grab useful info

Query Embedding: $u = Bq$ ← question

Attention: $P_i = softmax(u^T m_i)$

Output: $o = \sum_i P_i c_i$

Prediction: $\hat{a} = softmax(W(o + u))$

Model trained e2e on series of assertions/questions; learns A, B, C, W

Position Encoding allows embedded words to carry info about their location in the input

Memory Networks - using KV store ~ computed directly from context

- sentence level encoding: free text input broken into sentences, encoded in BoW as key, value = standard memNN
- window level - encode window of W words in BoW as key, use center word as value
- KB Triple: typically have form "subject relation object" - key is subject-relation pair, value is object
- original natural language ~~[illegible]~~ questions standardized using SimpleQuestions dataset

bAbI - given short simple sentences + question, produce one word answer

Scaling

- query text used to perform full-text search across DB - Soln (Prob: attn spread out, small grads)
- only documents similar enough to query are actually considered, which go into MemoryNet

use knowledge base when possible, fall back on free text

SQuAD 1.0 - Stanford Q/A Dataset

- 100K + questions posed by crowd workers on a set of Wikipedia articles · longer answers

SQuAD 2.0 - 1.0 + adversarial examples

- 1.0 could cheat by providing answer of appropriate type
- 2.0 includes unanswerable q's, preventing educated guesses

QANet

conv input + transformer

transformer encoder ↑ conv layers + linear } encoder block in nw

- uses LayerNorm to reduce covariate shift by scaling and biasing activations

Context Query Attention Block:

- calculate similarity between query, context embeddings
- row wise softmax ⇒ A

Query Context Attention Block:

- column wise softmax ⇒ B

Model Encoder blocks take in context, 2 attention matrices

[illegible] outputs softmax over word positions for start + end of answer span
- non recurrent design, training time doesn't grow w input length
- backtranslation provides more ~~depth~~ diversity in training data
- multiple attention heads
- bilinear combination of context + query

Goal-Directed Dialog Systems
- engage the user
- help user w goal-directed tasks
- learn from sample dialog how to respond to user queries

Task Summary
1) Issue API calls, w requested fields
2) Updating API calls - users update requests
3) display options retrieved from KB
4) Providing extra info
5) Conducting full dialogs

Datasets:
Restaurant Reservations:
- contains two KBs of 4200 facts, 600 restaurants each

Dialog State Tracking Challenge
- restaurant booking data from real users
- every dialog turn labeled with a state to be predicted

Online Concierge Service
- Data from real online concierge service: users make requests through chat interface handled by human operators who make API calls

BERT: language model trained on large dataset of natural language fine-tuned for particular tasks
- uses bidirectional transformer

Generative Pre-Training (GPT): pretrain on language modeling
- transformer-based generator using only simplified transformer decoder stage
- uses left-to-right transformer
- discards multi-head attention (middle) step
- initially trained on large text corpus to minimize language modeling loss

$L(u) = \sum_i \log P(u_i \mid u_{i-k} \dots u_{i-1}; \Theta)$

For each task, custom linear layer is added + entire network retrained
- can be applied to a variety of non-generative tasks
  - entailment tasks
  - Question answering + commonsense reasoning

GPT2 - depth 48 GPT trained on WebText

BERT - recover dropped words (pretraining task)
- bidirectional transformer encoder model
- trained with two types of loss
  1) Word Prediction: 15% of input words are removed then re-predicted
  2) Next Sentence Prediction - predict whether pairs of sentences are consecutive
     - = contrastive loss: contrasts true positives with "near miss" negatives

Input Representation:
token embeddings + segment embeddings + position embeddings

Tasks:
- Sentence Pair Classification Tasks
- Single Sentence Classification Tasks
- Question/Answering Tasks
- Single Sentence Tagging Tasks

Dialog System Technology Challenge
- Sentence Selection
- Sentence generation
- Audio-Visual scene-aware dialog
→ BERT performs best

CoQA: Dataset
127k + questions w answers from 8000 + conversations

BERT, GPT, ELMo = pre-training for downstream tasks, similar to word2vec

ELMo uses RNN: concat indep. trained L→R, R→L LSTMs to generate features for downstream

BiDirectional Encoder models (BERT) better than generative models (GPT) at non-gen tasks
- gen models have training efficiency + scalability advantages that may make them more accurate
- can solve entirely new kinds of tasks w text generation

Adversarial Examples
1) $\eta = \epsilon \, \mathrm{sign}(\nabla_x J(\Theta, x, y))$

sign of gradient = perturbation

2) ↗ makes bigger change than optimum

Optimization Based Attack:
- make as small as possible change
- make classifier output target class → maximize label loss

Perturbation must triumph messy physical world
- need to find common δ to perturb image
- take into account spatial constraints
- apply perturbation on only part of image using mask

Adversarial perturbations are possible in physical world under different conditions and viewpoints, including distances + angles

∃ different approaches to optimize objective
→ need to know model f_θ

Black-box Attacks (don't know model)

Zero Query
- Random Perturbation
- Difference of Means
- Transferability Based

model ensembles used to assist adversarial image attacks

- train example to fool source learning method, transfer to target method

Non-Targeted: get classifier to predict any incorrect label. easier.

Targeted: get classifier to predict target label. harder.

Targeted Adversarial Examples: transferability is POOR.
- create adversarial examples from white box model, transfer to black box system.

Query Based Attack
- Finite difference gradient estimation
- Query reduced gradient estimation
→ d-dimensional vector X, make 2d queries to estimate gradient

$$FD_x(g(x), \delta) = \begin{bmatrix} \frac{g(x+\delta e_1) - g(x-\delta e_1)}{2\delta} \\ \vdots \\ \frac{g(x+\delta e_d) - g(x-\delta e_d)}{2\delta} \end{bmatrix}$$

finite difference could be better than white box

$x_{adv} = x + \epsilon \cdot \mathrm{sign}(FD_x(\ell_f(x, y), \delta))$

For query reduced gradient estimation:
- random grouping - PCA
- random direction sample
  - gradients along some directions (dimensions) are weak and fall off
  - can find gradient in strongest dimension
  - only need very few queries to find this

use GAN to help w perturbation
- want to minimize distortion
1) train classifier to discriminate real/artificial images
   - get target label, minimize distortion
2) classifier classifies adversarial examples as well as possible
- discriminator loss replace L2 loss to get adversarial generator to produce "real" images

GAN attack relatively resilient against defenses

Geometric distortion flow - move than perturbing each individual pixel

Why do Adversarial Examples Work?
- shifts attention

inception_v3 model ≈ human perception
- can train models to resist adversarial examples

Adversarial trained models unable to discriminate GAN-produced examples

can fool auto encoders: input ⇒ latent code ⇒ output
- VAE-like generative models vulnerable to adversarial examples
- can also fool RL - perturbation on every frame (but not necessary)

Local Intrinsic Detection
- ∃ distribution of natural images (density)
- adversarial examples are outliers
- natural images have some set of neighbors

Generative Models
- represent full joint $p(x, y)$

Classifier: given x, y; determine $p(x, y)$

Generator: given y, generate sample x from $p(x, y)$

Auto Encoders
- transform input x into code variable z, back to synthetic input x'
- $\dim(z) \ll \dim(x)$ = info bottleneck

Encoder: very similar to, or is neural classifier

Code: representation of input designed to support reconstruction
- could be set of class labels, often encodes within-class variability

Decoder: conditional synthetic input generator (generative model)

Can create decoder as inverse of encoder - activation maximization

BUT: - inversion is iterative - slow
- train classifier to minimize loss $L(x, x')$
- $z \to x'$ not differentiable, KL

Implicit Auto Encoders
- suppose we have $x = f(z, \Theta)$
- invert $f(\cdot, \Theta)$ to get z and minimize $L(x, x' = f(z, \Theta))$
- slow bc iterative inversion
- slow bc high variance from individual samples

VAEs - L2 loss
- build decoder as probabilistic model $P_\Theta(x \mid z)$
- invert using variational inference to get approximation to $P_\Theta(z \mid x)$, $q_\phi(z \mid x)$

$z = g_\phi(x, \epsilon), \ \epsilon \sim N(0, 1)$

normal distr of z-values with mean g_0 and SD g_1

Loss Fn:
optimize marginal likelihood of data

$\sum_{i=1}^{n} \log P(x^i)$

Maximize variational lower bound:

$\mathcal{L}(\Theta, \phi; x) = E_{z \sim q_\phi(z \mid x)}[\mathcal{L}(\Theta, \phi; x, z)]$

Jensen's Inequality:
for concave $f(x)$, linear fn $\ell(x)$ tangent to $f(x)$ at $\bar{x} = E_p[x]$

$f(x) \le \ell(x)$

$\nabla = E_{\epsilon \sim p(\epsilon)}[\nabla_{\Theta, \phi} \mathcal{L}(\Theta, \phi; x, g(x, \epsilon))]$

Update parameters Θ, ϕ until conv.

Loss estimator has high variance
→ alternate estimator w KL divergence, which assumes that $q_\phi(z \mid x)$ allows density estimation

Simple Loss: $\hat{\mathcal{L}}(\Theta, \phi; x) \approx \frac{1}{L} \sum_{i=1}^{L} \log p_\Theta(x, z^i) - \log q_\phi(z^i \mid x)$

Θ, ϕ gradients are discrete

Alternative:

$\hat{\mathcal{L}}^B(\Theta, \phi; x) \approx -D_{KL}(q_\phi(z \mid x) \,\|\, P_\Theta(z)) + \frac{1}{L} \sum_{i=1}^{L} \log P_\Theta(x \mid z^i)$

Θ grad is approximate, ϕ is exact

BOTH: require $q_\phi(z \mid x)$ to provide a density and be integrable in closed form.

$z = g(x, \epsilon) = g_0(x) + g_1(x) \cdot \epsilon$

VAEs are efficient but have problems w blurriness
- say nothing about form of encoder + decoder fns

Autoregressive Models - full generative model (can compute density)

Key: pixels highly correlated over various scales
- generate one pixel at a time conditional on values of other pixels

Local Autoregressive
- examine only a few nearby pixels at each step

PixelCNN
- iteratively generate 1 pixel at a time
- fast to train - no recurrence
- slow at generating - pass through entire network for each pixel
- during training must use mask to ignore later pixels

PixelRNN - row LSTM
- uses recurrence instead of 3x3 conv to allow long range dependencies
- able to generate full row of pixels in 1 pass
- takes longer to train than PixelCNN (recurr)

Diagonal BiLSTM
- Row LSTM uses recurrent state only from triangular subset of previously computed states
- Diag BiLSTM uses all

Gated PixelCNN
- replace ReLUs w gating
- >> PixelRNN

Image Transformer
- autoregressive design (looks back)
- use multi-headed attn network
- use large influence region of image without recurrence
- computes pixel rep in layers
- uses dropout for regularization
- 3D pos encoding, now with color channel

GANs
- human compare generated image w image in dataset
→ 2 network architecture
  1) generates image
  2) nn discriminator to compare real + synthetic images
- GANs learn transformation from known simple distribution

Minimax: generator tries to increase discriminator loss, discriminator decreases
→ local Nash equilibrium found w adversarial training

Generator Network: generate realistic images
- learn transformation from random distribution to target distribution

Discriminator Network
- measure "goodness" of generated samples
- outputs probability of image being real

Step 1: look at generated images
Step 2: look at real images

Discriminator: maximize probability real classified real, minimize generated classified real

Generator: ↗ maximize generated = real

GANs actually produce distr of images similar to real distribution
- minimize Jensen-Shannon Divergence between $P_{data}(x)$, $x = G(z)$

Algorithm
For each training iteration:
  For k steps:
    Sample batch of m noise samples
    Sample batch of m examples from $P_{data}(x)$
    Update D by ascending GrD
  Update G by descending GrD

Maximize $\frac{1}{m} \sum_{i=1}^{m} \log(D(G(z^i)))$
bc other way: $\log(1 - \dots)$ gives small gradients when generator samples are bad
- want large gradients when generated samples are bad

In practice, must balance D/G to ensure ∃ good gradients
- GD on cost fns not guaranteed to converge to actual Nash equilibrium

DCGAN
- generation now from all conv net
- fractional-transposed conv
- replace pooling w strided conv
- batchnorm in D/G
- leaky ReLU in D
- remove FC layers
- use ReLU in G except tanh for out

Conditional GAN
- add label y to specify classes in dataset you want to sample from

ACGAN
- ask discriminator output class label

Text-to-Image Synthesis
- embedded y (label) is sentence vector
- supports interpolation on y

Problems:
Mode collapse ⇒
- experience replay: substitute old D/G every few iterations
- anticipate gradients D is taking, allow generator to look ahead

Difficulty in Training ⇒
- WGAN - Wasserstein Distance
- min cost to match distr.
→ can keep (many) until converg.
→ gradients everywhere

Difficulty in Understanding Global Struct
- Progressive GANs: build resolution
  - high resolution
  - add layers to GAN, add upsampling layers incrementally

BigGAN
~ resnet w lots of power
- upsample

Difficulty in Evaluation
- inception score
$P(y)$ = diversity, $P(y \mid x)$ = clearness