LeNet: visual classification network that recognized digits for zipcodes

Representation Learning: learn from input image/output labels
- early layer weights task independent

⇒ 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 ⇒ 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

softmax

Cross Entropy Loss: negative log probability 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

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

Variance - variance of predictions

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

Variance >> Bias²: too much variation between models. overfitting

Bias² >> 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: $v^T M v \ge 0 \; \forall v \in \mathbb{R}^n$
- all $\lambda_i \ge 0$

Regularized Multivariate Regression:

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

- lower variance, higher bias

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

Strong regularization (large λ)
⇒ lower variance, higher bias
generalizing model

Weak Regularization (small λ)
→ higher variance, lower bias

SVMs: large margins to give room for error
classify as many correctly as possible

- want to maximize classifier margin

$f(x) = w^T x + b = 0$ boundary

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

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

Hinge Loss: ~~[illegible]~~
↳ $\max(0, 1 - y f(x))$
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

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

Softmax: $f_j(z) = \frac{\exp(s_j)}{\exp(s_1) + \dots + \exp(s_k)}$

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

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

RMS Prop 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

B: [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 ↓ $1/\sqrt{t}$

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

ADAM: momentum + RMS Prop
- compute moving averages of gradient + squared gradient

$p^t = (B_1) p^{t-1} + (1 - B_1) g^{(t)}$

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

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

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 - FCNN 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×m filter $(n \cdot m \cdot \ell + 1)\,k$

ℓ input feature map - depth Conv

k output feature map - num filters

FC: n input, m output: $(n+1)\,m$

Convolutional Filters:
- assume nearby pixels share similar structure
- params for filter are shared across image

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

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

Input → low level feature → high level feature → trainable classifier

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

Image: N×N, Filter: F×F

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

Conv Layer:
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
· 5×5 conv filters at stride 1
[conv-pool]×2 - conv - fc 1998

AlexNet: similar architecture, bigger/deeper
1st layer: 96 11×11 filters w stride 4
output: [55×55×96]
In: [227×227×3]
~35K params
[conv-pool-norm]×2 - [conv]×3 - pool - [fc]×3
- first use of ReLUs, GPUs, dropout
~60 million params

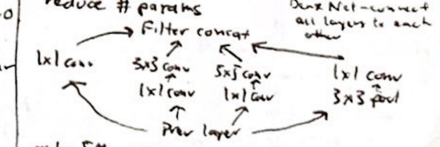
Transfer Learning
1) Train on ImageNet
2) - small dataset
   · retrain only classifier, i.e. last (end) softmax layer
   - medium dataset
     · perform fine tuning
       - used old weights as initialization, train full network or only some of higher layers
       - maybe retrain lower layers

VGGNet 2014
[conv-conv-pool]×2 - [conv-conv-conv-pool]×3 - FC×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 56×56
use skip conn to propagate faster thru nw

- fewer params than VCG
- 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
- non linearity
- 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
   - repeats 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
- ~~pool~~ 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

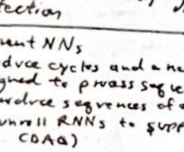
State of the Art: Single Shot Detection
- base boxes centered at each grid cell
Within each:
- regress from each of B base boxes to final box w 5 numbers (dx, dy, dh, dw, confidence)

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

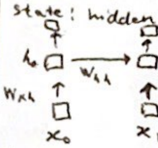
Recurrent NNs
- introduce cycles and a notion of time
- designed to process sequences of data + produce sequences of outputs
- can unroll RNNs to support backprop (DAG)

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

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 score 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
Excitatory
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: l+ fwd pass
R: after fwd pass
Backprop Grad.