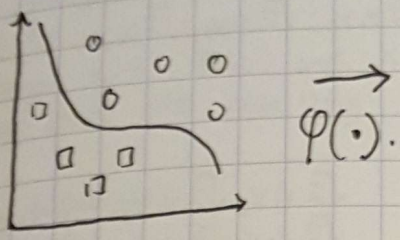
KERNELS (CSC411 NOTES VS SLIDES). (attempt red)

non-linear decision boundaries.

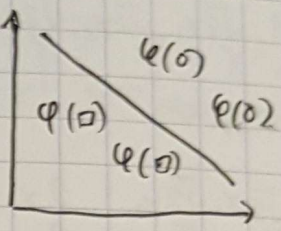
Feature mapping: Take input $x \to \phi(x)$.

INPUT SPACE:



$\varphi(\cdot)$.

FEATURE SPACE:



IF $x_i = [x^1, x^2, \ldots, x^d]$, d high = HIGH COMP. COST.

QUADRATIC FEATURE BOUNDARY IDEA:

For $x = (1, x_1, x_2, \ldots, x_d)$ we have:

$\phi(x) = (1, \sqrt{2}x_1, \ldots \sqrt{2}x_d, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \ldots \sqrt{2}x_{d-1}x_d, x_1^2 \ldots x_d^2)$

Example: $d = 3$

$x = (1, x_1, x_2, x_3)$.

$\phi(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_3, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \sqrt{2}x_2x_3, x_1^2, x_2^2, x_3^2)$.

NOTE: dimension of $\phi(x)$ is thus order $O(d^2)$.

KERNELS

Recall: for $x = (1, x_1, x_2, x_3)$,

$\phi(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_3, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \sqrt{2}x_2x_3, x_1^2, x_2^2, x_3^2)$.

Then: DEFINE K AS INNER PROD OVER $\phi(\cdot)$

So,

$K(x,y) = \langle \phi(x), \phi(y) \rangle$

$K(x,y) = \langle \phi(x), \phi(y) \rangle$

And,

$\langle \phi(x), \phi(y) \rangle =$

$\langle (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_3, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \sqrt{2}x_2x_3, x_1^2, x_2^2, x_3^2),$
$(1, \sqrt{2}y_1, \sqrt{2}y_2, \sqrt{2}y_3, \sqrt{2}y_1y_2, \sqrt{2}y_1y_3, \sqrt{2}y_2y_3, y_1^2, y_2^2, y_3^2) \rangle$

The cross product:

$1 \cdot 1 = 1$, $\sqrt{2}x_1\sqrt{2}y_1 = 2x_1y_1$, $\sqrt{2}x_2\sqrt{2}y_2 = 2x_2y_2$, $\sqrt{2}x_3\sqrt{2}y_3 = 2x_3y_3$

$\sqrt{2}x_1x_2\sqrt{2}y_1y_2 = 2x_1x_2y_1y_2$, $\sqrt{2}x_1x_3\sqrt{2}x_1y_3 = 2x_1x_3y_1y_3$, $\sqrt{2}x_2x_3\sqrt{2}y_2y_3 = 2x_2x_3y_2y_3$

Result: $K(x,y) = (1 + \langle x,y \rangle)^2$.

Computes K, $O(d)$ time/memory, K is polynomial

(WHY IS THIS RELEVANT LOL).

KERNELS: polynomial, gaussian, sigmoid

KERNEL PROPERTIES:

1) POS. CONST. FXN. IS A KERNEL:
FOR $\alpha \geq 0$, $K'(x_1, x_2) = \alpha$.

2) POS. WEIGHTED LIN COMBS. OF KERNELS ARE KERNELS:
$\forall i, \alpha_i \geq 0 : K'(x_1, x_2) = \sum_i \alpha_i K_i(x_1, x_2)$

3) PRODUCTS OF KERNELS ARE KERNELS
$K'(x_1, x_2) = K_1(x_1, x_2) K_2(x_1, x_2)$

KERNEL LEGOS: combine to make complex feature maps!

kernel feature spaces: become EXTREMELY complex, twisty, curvy!

KERNELS AND SVM (CS 229 NOTES)

$\gamma^i = y^i\left(\left(\frac{w}{\|w\|}\right)^T x^i + \frac{b}{\|w\|}\right)$

↑ margin.

Geom. Margin wrt (w,b) and x: $\gamma = \min \gamma^i$ over $i = 1 \ldots m$
For the m data pts resulting in $\gamma^1 \ldots \gamma^m$ margins.

OPTIMAL MARGIN.

1) optimization I:

maximize γ wrt γ, w, b
w/ constraints: $\|w\| = 1$ ← (ensures func marg = geom marg).
$y^i(w^T x^i + b) \geq \gamma \quad \forall i = 1 \ldots m.$

PROBLEM: $\|w\| = 1$ is a non-convex constraint.

2) Optimization II:

maximize $\hat{\gamma}/\|w\|$ wrt $\hat{\gamma}, w, b$ ($\hat{\gamma}$ = func. marg).
w/ constraints: $y^i(w^T x^i + b) \geq \hat{\gamma}, \quad \forall i = 1 \ldots m.$

PROBLEM: $\hat{\gamma}/\|w\|$ is a non-convex objective fxn.

3) optimization III:

Set $\hat{\gamma} = 1$. Then $\hat{\gamma}/\|w\| = 1/\|w\|$, which maximizing is the same as minimizing $\|w\|^2$. Then:

minimize $\frac{1}{2}\|w\|^2$
w/ constraints: $y^i(w^T x^i + b) \geq 1 \quad \forall i = 1 \ldots m.$

Solve w/ QP code or Lagrange duality.

LAGRANGE DUALITY

Consider minimize $f(w)$ wrt w
w/ constraint $h_i(w) = 0 \quad i = 1 \dots m$.

LAGRANGIAN:

$$\mathcal{L}(w, \beta) = f(w) + \beta_1 h_1(w) + \dots + \beta_m h_m(w)$$

β_i's = Lagrange Multipliers.

Solve for w and β via: $\frac{\partial \mathcal{L}}{\partial w_i} = 0 \;\; \forall i = 1 \dots m, \quad \frac{\partial \mathcal{L}}{\partial \beta_i} = 0 \;\; \forall i = 1 \dots m$

PRIMAL:

$$\min f(w) \text{ wrt } w.$$
$$\text{s.t. } g_i(w) \le 0 \quad i = 1 \dots n$$
$$h_i(w) \le 0 \quad i = 1 \dots m.$$

GENERALIZED LAGRANGIAN: $\sum_{i=1}^{n} \alpha_i g_i(w)$ $\qquad$ $\sum_{i=1}^{m} \beta_i h_i(w)$

$$\mathcal{L}(w, \alpha, \beta) = f(w) + \alpha_1 g_1(w) + \dots + \alpha_n g_n(w) + \beta_1 h_1(w) + \dots + \beta_m h_m(w)$$

Define $\theta_P(w)$ s.t.

$$\theta_P = \begin{cases} f(w) & \text{if } w \text{ satisfies primal constraints} \\ \infty & \text{else.} \end{cases}$$

where $\theta_P = \max \mathcal{L}(w, \alpha, \beta)$ wrt α, β, and $\alpha_i \ge 0$.

DUAL: $\theta_D(\alpha, \beta) = \min_w \mathcal{L}(w, \alpha, \beta)$

Then:

$$d^* = \max_{\alpha, \beta} \min_w \underbrace{\mathcal{L}(w, \alpha, \beta)}_{\theta_D(\alpha, \beta)} \le \min_w \max_{\alpha, \beta} \underbrace{\mathcal{L}(w, \alpha, \beta)}_{\theta_P(w)} = p^*.$$

THUS: WE CAN SOLVE DUAL IN LIEU OF THE PRIMAL.
we assume $d^* = p^*$.

Thus, maximizing dual $\leftrightarrow$ minimizing primal.

Thus, must exist w^*, α^*, β^* s.t. w^* soln of PRIMAL, α^*, β^* soln of DUAL, $p^* = d^* = L(w^*, \alpha^*, \beta^*)$, and w^*, α^*, β^* satisfy KKT conds:

$\alpha^* \geq 0$, $g_i(w^*) \leq 0$, $\alpha_i^* g_i(w^*) = 0$, $\partial_{\beta_i} \mathcal{L} = 0$, $\partial_{w_i} \mathcal{L} = 0$.

if $\alpha_i^* > 0$, $g_i(w^*) = 0$

OPTIMAL MARGIN CLASSIFIERS

Recall $\min \frac{1}{2}\|w\|^2$ wrt γ, w, b

s.t. $y^i(w^T x^i + b) \geq 1$.

Rewrite constraint as: $g_i(w) = -y^i(w^T x^i + b) + 1 \leq 0 \quad i = 1 \ldots m$.

NOTE: $\alpha_i > 0$ only for training examples s.t. their functional margin $= 1$, b/c they correspond to constraints $g_i(w) = -y^i(w^T x^i + b) = 0$.

They define SUPPORT VECTORS

Lagrangian for this optimization problem:

$$L(w, \alpha, \beta) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{m} \alpha_i \left(y^i(w^T x^i + b) - 1\right)$$

(No β_i b/c only inequality constraints on this optimal margin problem).

Taking derivative WRT w, b, we can get the dual problem wrt α.

KERNELS (CS 229)

Given "attributes" for each pt x, we can use a FEATURE MAPPING $\phi(x)$ to get nonlinear features.

From before: if we have found the α_i's, to make a prediction, we need to calculate:

$$w^T x + b = \left(\sum_{i=1}^{m} \alpha_i y^i x^i\right)^T x + b$$

$$= \boxed{\sum_{i=1}^{m} \alpha_i y^i \langle x^i, x \rangle + b.}$$

For our new $\phi(x)$ features, we just replace each x^i, x w/ $\phi(x^i), \phi(x)$. THUS,

$$w^T \phi(x) + b = \sum_{i=1}^{m} \alpha_i y^i \langle \phi(x^i), \phi(x) \rangle + b$$

DEFINE: KERNEL: $K(x,z) = \langle \phi(x), \phi(z) \rangle = \phi(x)^T \phi(z)$.

$\phi(x)$ might be hard to calculate, but $K(x,z)$ easy to calculate. e.g. $\phi(x)$ is EXTREMELY HIGH DIMENSION.

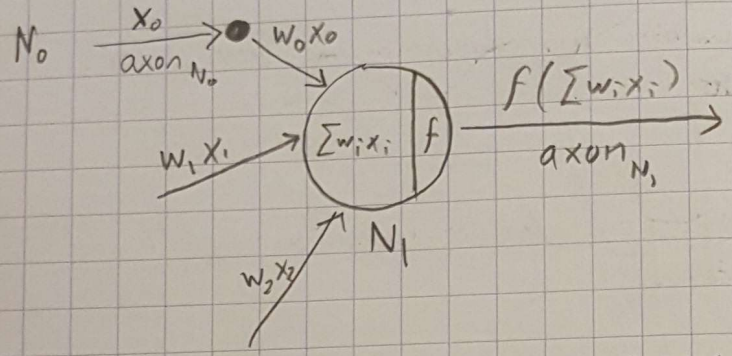
By calculating $K(x,z)$ efficiently, we can get an SVM to learn in ϕ w/o ever calculating $\phi(x)$ which would be hella inefficient.

* MULTICLASS CLASSIFICATION. - REVIEW.
* CH 4 → LINEAR MODELS FOR CLASSIFICATION.

NEURAL NETWORKS I

+ nonlinear discrimin. clsfrs using LARGE COLLECTIONS OF SIMPLE FUNCTIONS composed together.

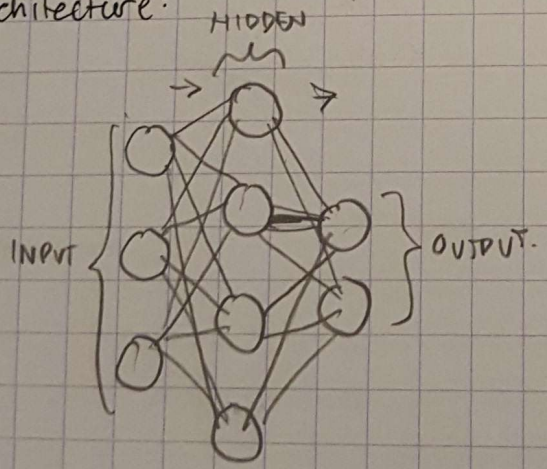
Diagram of a neuron:



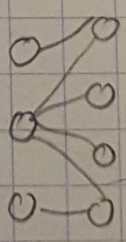
+ The dendrite inputs are "HIDDEN FEATURES".
+ common activation functions:

+ sigmoid: $\sigma(z) = \frac{1}{1+\exp(-z)}$
+ tanh: $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
+ ReLU: $\text{ReLU}(z) = \max(0, z)$.

Architecture:



+ Here, 4 hidden-units
+ OUTPUT OF UNIT: LINCOMB of outputs from its inputs, with an activation function applied on it
+ NOTE:



is possible, aka we don't need all unit inputs from $i-1^{th}$ layer for all units in y^{th} layer.

+ NN LAYERS. We a
REPRESENTATIONA
+ NN's are
+ Eg, $f_1(x)$ and
+ KEY: the
+ An NN w a univer discrimi
+ ↑ LAYE run th
PASSES:
+ FORW
+ BACK
FORWARD
+ LEG
+ OU
+ E
The

- NN LAYERS: We EXCLUDE the input layer. So the prev. arch. was a 2-LAYER NN.

REPRESENTATIONAL POWER!

- NN's are based on repeated, chained function composition.
- Eg: $f_1(x)$, $f_2(x)$ and the output of N_3 is $f_3(w_1 f_1(x) + w_2 f_2(x))$, and f_3 may be the input for another neuron.
- KEY: the weighting of the inputs (in the dendrites).

- An NN w/ ONE HIDDEN LAYER can represent ANY FUNCTION as a universal approximator. Thus, good base for nonlinear discriminants.

- ↑ LAYERS, ↑ HIDDEN UNITS = ↑ network capacity, but can run the risk of overfits.

PASSES:

- FORWARD PASS: Make the prediction.
- BACKWARD PASS: Compute the gradient.

FORWARD PASS: COMPUTING A PREDICTION ON x.

- LEGEND:
  - j will index hidden units
  - k will index output units
  - D is num inputs.
  - V_{ji} is a weight from INPUT i to HU_j.
  - W_{kj} is a weight from HU_j to OUT k.
  - $h_j(x)$ = OUT of HU_j
  - $O_k(x)$ = OUT of out k.

- OUTS:
$$h_j(x) = f\left(V_{j0} + \sum_{i=1}^{D} x_i V_{ji}\right).$$
$$O_k(x) = g\left(w_{k0} + \sum_{j=1}^{J} h_j(x)\, w_{kj}\right).$$

- Example: $x = (x_1, x_2, x_3)$, $J = 4$, $O = 2$

Then,
$h(x) = (h_1(x), h_2(x), h_3(x), h_4(x))$
$O(x) = (O_1(x), O_2(x))$

$h_j(x) = f(v_{j0} + x_1 v_{j1} + x_2 v_{j2} + x_3 v_{j3})$
$O_k(x) = g(w_{k0} + h_1(x) w_{k1} + \cdots + h_4(x) w_{k4})$

Thus, O, h, x are vectors!

+ IF we don't use an activ. fxn, it's just linear classification b/c we are only looking at linear combinations. (More explicitly: linear regression).

+ SINGLE LAYER (NO HIDDENS) + SIGMOID = this is just LOGISTIC REGRESSION

BACKWARD PASS: TRAINING THE NN VIA BACKPROP + ∇s.

+ Usually to optimize a classifier, we find **W** weight vector such that this weight vector minimizes our chosen loss function.

+ BUT: hidden units = objective fxn NO LONGER CONVEX. Empirically we know that many local minima are good though so we can still try SGD.

+ Enter BACKPROP. For each input **X** we need to do our classifying forward pass, THEN PROPAGATE GRADIENTS BACK and update **W**s via gradient descent.

BACKWARD PASS AND CHAIN RULE:

+ PROBLEM: when optimizing **W**, we need targets (aka test set). But hidden units don't give us targets, only output units do. WYD?

  + Compute HOW FAST ERROR CHANGES AS ACTIVITY CHANGES.
  + We use ERROR DERIVATIVES WRT HIDDEN ACTIVITIES
  + we need to COMBINE THE EFFECTS each hidden activity has on the error.
  + Once we have error derivatives for all HAs, we can get error derivs. for the weights entering a HU.

↳ CHAIN RULE.

$\partial sig = \sigma(z)(1-\sigma(z))$
$\partial tanh: 1/\cosh^2(z)$
$\partial reLU =$ 1 if $z>0$, 0 if $z \leq 0$.

SINGLE LAYER NET ∇s:

- ERROR GRADIENTS: $\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial z_k} \frac{\partial z_k}{\partial w_{ki}}$ where z_k is the COMBINED INPUT to o_k BEFORE THE ACTIVATION FXN.

Some notation: $\delta_k^o = \partial_{o_k} E$

$\delta_k^z = \delta_k^o \cdot \partial_{z_k} o_k$.

And we know: $\partial_{w_{ki}} z_k$ is just x_i, so $\partial_{w_{ki}} E = \delta_k^z \cdot x_i$.

CONCRETE EXAMPLE:

+ Let E be MSE and g be LOGISTIC SIGMOID A.F
+ Then...

$\partial_{o_k} E = \partial_{o_k} \frac{1}{2} (o_k^{(n)} - t_k^{(n)})^2 = o_k^{(n)} - t_k^{(n)}$ for example (n).

So $\delta_k^o = o_k^{(n)} - t_k^{(n)}$

Next, we know $o_k^{(n)}$ is logistic sigmoid, so

- $o_k^{(n)} = g(z_k^{(n)}) = (1 + \exp(-z_k^{(n)}))^{-1}$

$\partial_{z_k} o_k = o_k^{(n)} (1 - o_k^{(n)})$.

So, $\delta_k^z = \delta_k^o \cdot \partial_{z_k} o_k = (o_k^{(n)} - t_k^{(n)}) o_k^{(n)} (1 - o_k^{(n)})$

AND OUR FULL GRADIENT for $x^{(n)}$ is:

$$\partial_{w_{ki}} E = \sum_{n=1}^{N} (o_k^{(n)} - t_k^{(n)}) o_k^{(n)} (1 - o_k^{(n)}) x_i^{(n)}$$

AND we recall the GRAD UPDATE RULE:

$$w_{ki} \leftarrow w_{ki} - \eta \partial_{w_{ki}} E$$

EXAMPLE WITH A HIDDEN LAYER:

+ Legend:
  - Let O_k be the outputs
  - Let z_k be the combined linear combo inputs to out unit k
  - Let g be the output AF
  - Let w_{kj} be the weight of HU j on OUT k.
  - Let h_j be the HU outputs
  - Let u_j be the combined linear combo inputs to HU j.
  - Let f be the HU AF.
  - Let v_{ji} be the weight of INPUT i on HU j
  - Let x_i be INPUT i.

We can treat the $\partial_{w_{kj}} E$ as our single layer. But, instead of x_i, it's now h_j. Thus:

$$\partial_{w_{kj}} E = \sum_{n=1}^{N} \underbrace{\partial_{O_k^{(n)}} E \, \partial_{z_k^{(n)}} O_k^{(n)}}_{\delta_k^{z,(n)}} \underbrace{\partial_{w_{kj}} z^{(n)}}_{h_j^{(n)}}$$

$$= \sum_{n=1}^{N} \delta_k^{z,(n)} h_j^{(n)}$$

What about hidden weight gradients?
We use BACKPROP: FOR A SINGLE EXAMPLE $x^{(n)}$, calc $\partial_{h_j^{(n)}} E$:

$$\partial_{h_j^{(n)}} E = \sum_{k=1}^{K} \partial_{O_k^{(n)}} E \, \partial_{z_k^{(n)}} O_k^{(n)} \, \partial_{w_{kj}} z_k^{(n)} = \sum_{k}^{K} \delta_k^{z,(n)} w_{kj}$$

Call $\sum_{k}^{K} \delta_k^{z,(n)} w_{kj} = \delta_j^{h,(n)}$

And Finally...

$$\partial_{v_{ji}} E = \sum_{n=1}^{N} \underbrace{\partial_{h_j^{(n)}} E}_{\delta_j^{h,(n)}} \underbrace{\partial_{u_j^{(n)}} h_j^{(n)}}_{f'(u_j^{(n)})} \underbrace{\partial_{v_{ji}} u_j^{(n)}}_{x_i^{(n)}} = \sum_{n}^{N} \delta_j^{h,(n)} f'(u_j^{(n)}) x_i^{(n)}$$

MULTIPLE HIDDEN LAYERS

+ Use the same idea as over there. For layer L, we need to backprop: compute $\partial_{h_j^L} E$ and we can use it to compute $\partial_{v_{ij}^L} E$ (weights at layer L) and $\partial_{h_j^{L-1}} E$.

+ Training NNs is HARD: long compute times, vanishing gradient problem (the gradient will be vanishingly small, preventing the weight from changing its value, which can even stop the NN from more changing).

+ ReLU is a solution to van. grad, and tanh/sigmoid can saturate causing vanishings. However, neurons can still "die" w/ ReLU...

+ OUTPUT g: either sigmoid/softmax, (classification), or NO OUTPUT (regression).

+ WEIGHT INITIALIZATIONS:
  + CONSTANT → all neurons will stay same
  + STANDARD → $W_{ij} \sim N(0, \sigma^2)$: $\sigma^2 \downarrow$, converge to 0, $\sigma^2 \uparrow$, diverge.
  + XAVIER → $\sigma^2 = 2/n_{in} + n_{out}$
    n_{in}, n_{out} # of units in prev layer and next layer
  + HE → $\sigma^2 = 2/n_{in}$.
    Rec'd for ReLUs.

} choosing σ^2

+ NEED MOMENTUM: $v_{t+1} = \beta v_t + \nabla L(w_t)$. $\beta = 0.9$.
$x_{t+1} = x_t - \alpha v_{t+1}$.

NEURAL NETWORKS II.

+ some reasons why computers SUCK at scene/obj recognition:
  OCCLUSION, SCALE, DEFORM, CLUTTER, LIGHTING, VIEWPT, POSE.
  HUGE IN-CLASS VARIATIONS
  TONS OF CLASSES.

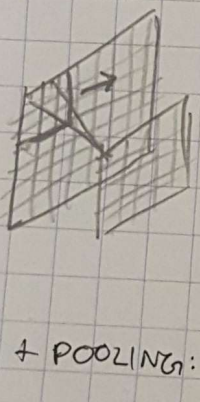
NEURAL NETS + IMAGES.

PREVENT OVERFITTING
+ Limit hidden unit #
+ Limit weight size (weight decay)
+ Early stopping
+ Dropout

+ LOCALLY-CONNECTED LAYERS: images have many pix, data ✗ is high dim. Fully-connected layers infeasible. Good when input image is registered. (faces). statistics similar at diff locs.
  + DIFFERENT FILTER FOR EACH (x,y) POSITION. 4M PARAMS...

+ REPLICATED FEATURE APPROACH (monkey vision):
  + Many diff copies of same feature detector.
  + Use several DIFF FEAT TYPES each w/ own replicated pool of detectors.

+ CNN: connect each hidden unit to SMALL INPUT PATCH and SHARE WEIGHT ACROSS SPACE.



200×200 image, 10×10 filters, 100 filters, 10K params

CNN hyperparams: DEPTH (# filters)
STRIDE (units apart a filter is applied)
SIZE (w×h) of filters.
Need to pad output to prevent shrinkage.

+ POOLING: MAX or AVG. of features at specific locations.

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool 2×2 filters stride=2 →

6	8
3	4

+ FILTERBANK → CUBE. Each slice is the output of convolving img w/ one filter. Apply activation fxn on each hidden unit.
+ MAX POOLING (robustness) (INVARIANCE IN SMALL POSITION SHIFTS).

+ # HIDDEN Us smaller than input or output units = not generalized, lost info. Keep them bigger! (can be used 4 autoencoders).

PRINCIPAL COMPONENTS ANALYSIS.

SOME GOALS OF UNSUPERVISED LEARNING
+ reducing data dimensionality (while retaining its key nature)
+ finding CLUSTERS (data w/ similarities, forming classes)
+ modelling data density
+ finding hidden causes.

1. DIM RED → rep each input case w/ small number of vars
2. CLUSTER → rep each input case w/ PROTOTYPE EXAMPLE
3. DENSITY EST → estim PROB DIST over the data

+ AIM OF PCA: find small # OF "DIRECTIONS" (axes) in input space that EXPLAIN INPUT DATA VARIANCE.
We then RE-PROJECT DATA ONLY ON THOSE AXES.
+ Assume data is continuous
+ CAN REDUCE OVERFITS
+ Good for visualization, preprocessing, lossy compression, better generalization.

Let $K \ll D$, and $\mathbf{z} \in \mathbb{R}^K$, so goal: $\mathbf{x} \approx U\mathbf{z} + \mathbf{a}$ and we project $\mathbf{x}$ to lower dimensions. U is a $D \times K$ MATRIX.

+ SEARCH for [ORTHOG DIRS IN SPACE w/ HIGHEST VAR].

+ CENTER the data. (FIND $\bar{x}$, subtract $x - \bar{x}$)
+ CALCULATE EMPIRICAL COVARIANCE MATRIX:

$$\Sigma = \frac{1}{N} X^T X \quad \text{Note: } X^T X = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \end{bmatrix} \begin{bmatrix} A_{11} & A_{21} & A_{31} \\ A_{12} & A_{22} & A_{32} \end{bmatrix}$$

And: result: 3 x 3 matrix
w/ elements: $A_{11}A_{11} + A_{12}A_{21}$ $A_{11}A_{21} + A_{12}A_{22}$ ...
etc.

+ Look for a direction/axis w that MAXIMIZES PROJECTION VARIANCE $y^i = w^T x^i$. Normalize: $\|w\| = 1$.

+ $\text{Var}(y) = w^T \Sigma w = \sum_i \frac{1}{N} (w^T x^i)^2 = \frac{1}{N} \sum_i w^T x^i x^{iT} w$

+ So, we want to maximize the variance and find the direction that maximizes variance. Thus:

$$w^* = \underset{\|w\|=1}{\text{argmax}} \; w^T \Sigma w.$$

HOW TO MAXIMIZE AND FIND w^*:

1) Σ has an EIGENDECOMP. w/ ORTHONORMAL BASIS VECTORS
2) Call them $v_1 \dots v_d$. and $\lambda_1 \geq \dots \geq \lambda_d \geq 0$.
3) Write w in terms of those bases:

$$w = \sum_i a_i v_i \quad \text{and} \quad \sum_i a_i^2 = 1.$$

4) This transforms our objective fxn.
Previously we wanted to maximize

$$w^T \Sigma w.$$

But b/c of eigendecomp now we need to maximize

$$\sum_{i : a_i^2 = 1} a_i^2 \lambda_i$$

TOP k DIMENSIONS ARE TOP k EIGENVECTORS

Another way to see:

+ $\Sigma = U \Lambda U^T$ and $U^T U = U U^T = I$
+ Take $z^i = U_u^T x^i$ Take first k eigenvectors
+
+ Therefore, $z = U_{1:k}^T (x - m)$ (where m is mean)
+ To project, we calculate empirical cov matrix and find its eigenvectors. That's PCA in a nutshell.
+ Approximate reconstruction of $\tilde{x} = U_{1:u} z + m$

FINDING Ws w/ Lagrange multipliers:

1) FINDING w_1:

GOAL: Maximize $w_1^T \Sigma w_1$ and $\|w_1\| = 1$

Let α be a Lagrange multiplier. And $w_1^T w = 1 \rightarrow 0 = 1 - w^T w$

Then:

$$w_1^T \Sigma w_1 + \alpha(1 - w_1^T w_1)$$

Take derivative w.r.t w_1:

$$\Sigma w_1 + \alpha w_1 = 0.$$
$$\Sigma w_1 = \alpha_1 w_1 \quad \text{so } w_1 \text{ is eigenvector of } \Sigma \text{ w/ eigval } \alpha_1.$$

This also shows w_1 must be the eigenvector w/ max eigenval of max value b/c this maximizes objective

2) FINDING w_2

GOAL: Maximize $w_2^T \Sigma w_2$ w/ $\|w_2\| = 1$ and $w_2^T w_1 = 0$

LAGRANGE:

$$w_2^T \Sigma w_2 + \alpha_2 (1 - w_2^T w_2) - \beta w_2^T w_1.$$

Derivative wrt w_2:

$$\Sigma w_2 - \alpha_2 w_2 - \beta w_1 = 0$$
$$w_1^T \Sigma w_2 - \alpha_2 w_1^T w_2 - \beta w_1^T w_1 = 0$$
$$\alpha_1 w_1^T w_2 - \alpha_2 w_1^T w_2 - \beta w_1^T w_1 = 0$$
$$\alpha_1 0 - \alpha_2 0 - \beta = 0.$$
$$\beta = 0$$

Therefore, $w_2^T \Sigma w_2 + \alpha_2 (1 - w_2^T w_2)$ results in

$$\Sigma w_2 = \alpha_2 w_2.$$

y_i = dependent variable
NTINUOUS OUT-

have =
$\hat{y}) = 1$
$f_w(x,$

t-SNE + STOCHASTIC NEIGHBOUR EMBEDDING

+ PCA tries to find GLOBAL STRUCTURE so its bad for fine details, can lead to local inconsistencies (for pts = nearest neighbour)

+ t-SNE FINDS LOCAL STRUCTURE.
But is almost only used for visualization as a result b/c can't easily embed new pts.

SNE BASIC IDEAS

+ Encode the HDim data as distro, do a "random walk thru data, ↑ prob to jump to close pts, and find Low Dim pts w/ sim neighbourhoods
+ Distribution distance: KL distance
+ E.g. Gaussian distro. x_i pt, choosing x_j pt will be prob less as farther away

$$P_{j|i} = \frac{\exp(-\|x^i - x^j\|^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x^i - x^k\|^2/2\sigma_i^2)}$$

prob of choosing pt x_j from x_i , $P_{i|i} = 0$.
$\sigma_i \rightarrow$ sets size of neighbourhood
$\sigma_i \rightarrow$ diff for each x_i

+ Final distro, symmetrize over pairs: $P_{ij} = \frac{1}{2N}(P_{i|j} + P_{j|i})$

+ PERPLEXITY: $perp(P_{j|i}) = 2^{H(P_{j|i})}$
$H(P_{j|i}) = -\sum_k P_{j|i_u} \log(P_{j|i_u})$
P uniform over k elems = perp is k.
Low perp = ↓ σ^2
High perp = ↑ σ^2

Perplexity: How well prob distro predicts sample
5-50 good.

+ When we h
Q_{ij} Q

Optimiz
$y^1, y^2 \ldots$

KL(Q||P)
expect
KL(Q||
KL(Q

+ SNE

t-

Q_{ij}

∂_x

+ When we have distro P w/ P_{ij} we can now define Q of Q_{ij}

$$Q_{ij} = \frac{\exp(-\|y^i - y^j\|^2)}{\sum_k \sum_{l \neq k} \exp(-\|y^l - y^k\|^2)}$$

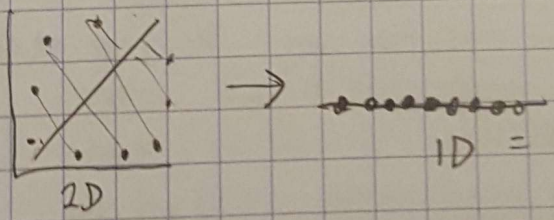
Optimizing Q close to P means minimizing KL-divergence.
$y^1, y^2 \ldots y^n$ are embeddings we are optimizing

$$KL(Q\|P) = \sum_{ij} Q_{ij} \log\left(\frac{Q_{ij}}{P_{ij}}\right)$$

distribution distance.

$KL(Q\|P)$ is the "penalty" for using the wrong distro of expecting P but getting Q. $KL(Q\|P) \geq 0$, $=0$ when $Q=P$.
$KL(Q\|P)$ is CONVEX. IF $P_{ij}=0$ but $Q_{ij} > 0$, then $KL(Q\|P) = \infty$ (VERY BAD)

+ SNE: We have P.
Looking for $y^1 \ldots y^N \in \mathbb{R}^D$ s.t. our inferred distro Q will MINIMIZE $L(Q) = KL(P\|Q)$.
Can show $\partial_{y^i} L = \sum_j (P_{ij} - Q_{ij})(y^i - y^j)$
Not convex.
PROBLEM: CROWDING: As we embed in lower dimensions, less space, shorter distances:



t-SNE: change SNE Gaussian of Q to a t-DISTRIBUTION (heavy-tailed Gaussian)
Slower change in Q → more "room"

$Q_{ij} = (1 + \|y_i - y_j\|^2)^{-1}$ instead of $\exp(-\|y_i - y_j\|^2)$.
w/ t-Distro.
$\partial_{y^i} L$ = same but + mes $(1 + \|y_i - y_j\|^2)^{-1}$.

CLUSTERING.

+ Grouping N examples into K clusters.
+ We initially assume the data belongs to K classes.
+ We assume data pts from the same class are similar (close Euclidean dist).

+ K-MEANS: start randomly w/ K centers and try using
  + Initialization
  + ASSIGNMENT STEP: Each data pt assigned to closest mean
  + REFITTING STEP: Adjust the means:

$$m_k = \frac{\sum_n r_k^{(n)} x^n}{\sum_n r_k^{(n)}}$$

$r_k^{(n)}$ = responsibilities

ENSEMBLES.: A return to supervised learning.

+ ENSEMBLE OF PREDICTORS: a set of predictors whose INDIVIDUAL DECISIONS combined to CLASSIFY NEW EXAMPLES
  1. Generate MULTIPLE CLASSIFIERS
  2. Each votes on TEST INSTANCE.
  3. Take MAJORITY/AVERAGE AS PREDICTION.

+ BAGGING: train SEPARATE MODELS on OVERLAPPING TRAIN SET. Average predictions
+ BOOSTING: Sequential, ITERATIVE re-weigh of training examples so current classifier can focus on HARD EXAMPLES

BIAS-VARIANCE DECOMPOSITION.

Error = (bias)^2 + variance + noise

$$\mathbb{E}_t[(t - y(x;D))^2] = \mathbb{E}_t[(t - h^*(x) + h^*(x) - y(x;D))^2] =$$

$$\mathbb{E}_t[(t - h^*(x))^2] - 2\mathbb{E}_t[(t - h^*(x))(h^*(x) - y(x;D))] + \mathbb{E}_t[(h^*(x) - y(x;D))^2]$$

noise. bias 2

$$\mathbb{E}_D[(h^*(x) - y(x;D))^2] = \mathbb{E}[(h^*(x) - \mathbb{E}[y(x;D)])^2] + \mathbb{E}_{y,x}[(y(x;D) - \mathbb{E}[y(x;D)])^2]$$

variance

+ BAGGING: Sampling w/ replacement → used for data paucity.
+ Random Forest. Goal = reduce variance, reduce overfits

+ OOB ESTIMATION: predict each training example using ALL TREES that DID NOT CONTAIN THIS in training data

+ Even if a single model is great, used in most competition winners.
+ Does not reduce bias. Need more randomness (random forest does this). OOB reduces the need of cross-validation.

BOOSTING.
+ BIAS reduction method in contrast with BAGGING which reduces variance
+ ADABOOST (adaptive boosting): Take a "weak" classifier doing ε better than chance, "BOOST" to get low training error

CLASSIFIER: $H(x) = \text{sign}(\sum_i \alpha_i h(x_i))$.
where h_i are weak classifiers
SUM OF LINEAR CLASSIFIERS NOT LINEAR

+ ADABOOST IDEA: Each iteration, reweigh training sample. (↑ weight to points wrongly classified)
Train new weak classifier, minimizing weighted acc.

+ ALGORITHM:
1) INPUT $\{x^n, t^n\}_{n=1}^N$, example weights $D_n^m(x) = 1/N$
2) $m = 1 \to M$:
a) $h_m(x)$ = weak learn x, t, w
b) $F_1 + h_m(x)$ by minimizing

$$J_m = \sum_{n=1}^{N} D_m^n [h_m(x^n) \neq t^n]$$

c) weighted error rate, classifier coefficient:

$$\epsilon_m = J_m / \sum D_m^{(n)}$$
$$\alpha_m = \tfrac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m}$$

d) UPDATE: $D_{m+1}^n = D_m^n \exp(-t^n \alpha_m h_m(x^n))$