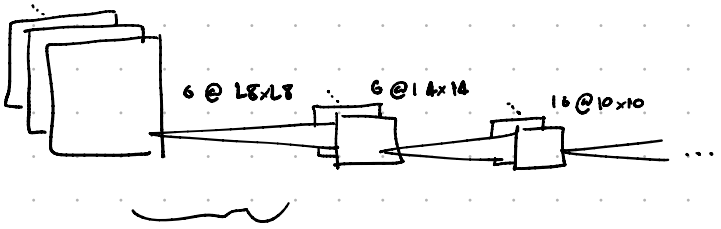
Padding: → zero padding controls the output size.
same convolution.
→ valid-only convolution: output only if the entire kernel contained in input.

Is there change in trainable parameters? / number of operations
→ yes, more operations.

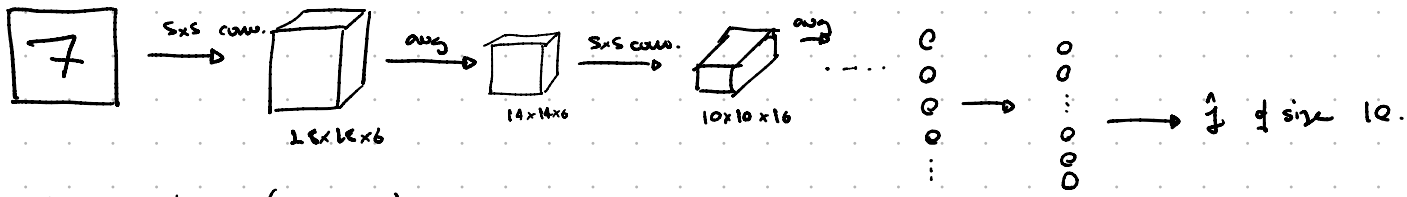
This is powerful for images, 2D data, speech.

LeNet-5: 3 convolutional layers (different hierarchies of features)



What does pooling do? Some sort of dimensionality reduction.
→ max, avg, etc...

6, 5x5 filters
= 150 trainable parameters.



→ ⃗ of size 10.

Output size: $\frac{(N+2P-F)}{\text{stride}} + 1$.

AlexNet: deeper in terms of layers, conv. + ReLU.

227 x 229 x 3 (224 before padding)

First layer: 96 filters, size 11x11, stride = 4

⇓

5 convolutional layers, size of filters important.

Equivalence between point-wise & something.

Usually NNs have been trained before transfer learning. AlexNet on ImageNet.

AlexNet(pretrained) + fine-tuning on new dataset

D_1 → output. p_0

D_2 [freeze |] → performance p_1
train

D_3 → train from scratch. p_2

is there a case where $p_2 > p_1$?
Transfer learning is popular for a reason.
↳ Dimensions that contribute to the classification.

Right now, every data point is completely independent. We do not assume any dependencies between samples.

Dropout gives you more regularization → equivalent to L2.

VGG Net: deep, better performance. Gives us hierarchical rep. of visual data to work.
↳ smaller filters, deeper.

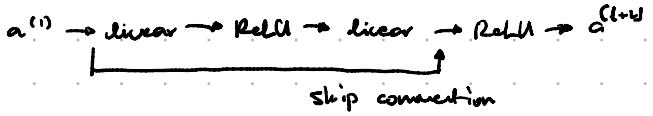
For now, same filter size even though maybe different init.
What if we vary filter size & then concatenate?
→ This is Inception models.

GoogleNet: no fully connected layers.
No more stacking of conv. + pooling layers.

The idea of vanishing gradients. For deep NNs.

Skip connections?? → helps w/ vanishing gradient.

ResNet uses skip connections.

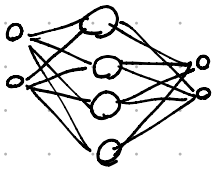


What happens if we just keep stacking?
Deeper models perform worse, but do not overfit.

ResNet 152 layers, but better performance.
Mechanism of skip connections not explained.

Right now, we deal with one-one. Input → output.
many-one. Inputs → output
one-many Image + caption
many-many (RNNs)

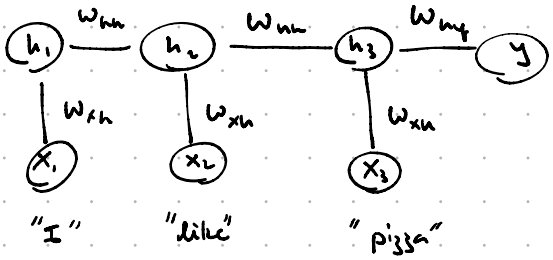
FFNs:



} Limitation: no "memory" of past, since weights learned independently → many parameters.

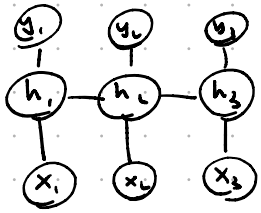
] Solution: Recurrent neural nets. (RNNs)

RNNs:

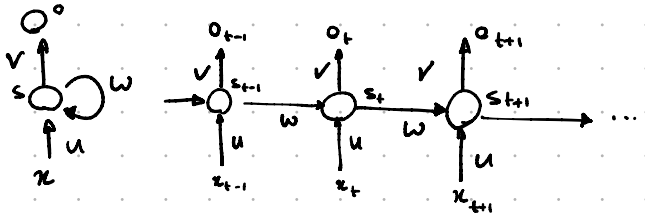


computational efficiency → weights are shared.

many-many



Many-one



The model shares the parameters (W, U, V). The things that are different are $s_{t-1}, s_t, s_{t+1}, \dots$
$x_{t-1}, x_t, x_{t+1}, \dots$

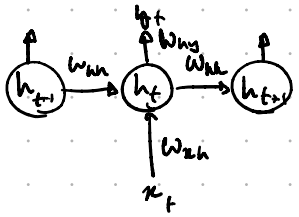
In lecture he uses h_t instead of s_t.

$h_t = \sigma(W_{xh} x_t + W_{hh} h_{t-1} + b)$ ← simplest RNN.
↳ activation function.

$y_t = W_{hy} \cdot h_t$ [might have softmax, etc... or linear]

If we get to h_{t+100}. Backprop, the intuition is it needs to propagate all the way. The longer your data, the worse (longer sequence). ⇒ vanishing gradient possible

$W = T \Sigma T^{-1}$
$W^n = T \Sigma^n T^{-1}$
} eigenvalue decomp.
↳ if $\lambda > 1$, exploding.
$\lambda < 0$, vanishing



$h_t = \tanh[W_{xh} x_t + W_{hh} h_{t-1} + b]$
current input, previous output

$\sum_{a=t}^{T} \frac{\partial l_a}{\partial W_{hh}}$

$\Rightarrow \frac{\partial l_T}{\partial y_T} \frac{\partial y_T}{\partial h_T} \boxed{\frac{\partial h_T}{\partial h_t}} \frac{\partial h_t}{\partial W_{hh}}$

↑ interested in this.

Recording from Nov. 4th: (part 1)
→ continuation of RNN

$\frac{\partial h_T}{\partial h_t} = \frac{\partial h_{t+1}}{\partial h_t} \cdots \frac{\partial h_{k-1}}{\partial h_{k-2}} \frac{\partial h_T}{\partial h_{t-1}}$

Assume $T = 0$; $t = 0$:

$\Rightarrow \frac{\partial h_1}{\partial h_0} \frac{\partial h_2}{\partial h_1} \frac{\partial h_3}{\partial h_2} \frac{\partial h_4}{\partial h_3} \frac{\partial h_5}{\partial h_4} = \frac{\partial h_5}{\partial h_0}$

$$h_{t+1} = \tanh\left[W_{xh} x_{t+1} + W_{hh} h_t + b\right]$$

$$\frac{\partial h_{t+1}}{\partial h_t} = \frac{\partial}{\partial h_t}\left[\tanh\left(W_{xh} x_{t+1} + W_{hh} h_t + b\right)\right]$$

$$= \text{diag}\left[\tanh'\left(W_{xh} x_{t+1} + W_{hh} h_t + b\right) \cdot W_{hh}\right]$$

$$W_{hh} = T \Sigma T^{-1}$$

$$W_{hh}^{\alpha} = T \Sigma^{\alpha} T^{-1}$$

exploding / vanishing gradient depending on the Σ^{α}.

We can solve exploding gradient problem by clipping the gradients.

↳ Vanishing is more common.

Let's change the way RNN works to solve vanishing gradient.

↳ Gates: → update gate: long term dependencies → $z_t = \sigma(W_{xz} x_t + W_{hz} h_t + b_z)$

↑ $0 \text{ or } 1 \Rightarrow$ sigmoid.

→ reset gate: short term dependencies.

↳ $r_t = \sigma(W_{xr} x_t + W_{hr} h_{t-1} + b_r)$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \tanh\left(W_{xh} x_t + r_t \odot (W_{hh} h_{t-1}) + b_h\right)$$

- Let z_t be 1: $\Rightarrow h_t = h_{t-1}$. There is no effect from x_t. Get rid of that if $z_t = 1$. [update gate]
- Let $r_t = 0$: $z_t \odot h_{t-1} + \tanh\left[W_{xh} x_t + b_h\right]$
- Let $z_t = 0$: $\tanh\left[W_{xh} x_t + r_t \odot (W_{hh} h_{t-1}) + b_h\right]$

↑ what if $r_t = 1$?

$\Rightarrow \tanh\left[W_{xh} x_t + W_{hh} h_{t-1} + b_h\right]$

This is the usual RNN.

Extreme cases.

By setting z_t & r_t, control the memory factor.

Why do we do this? Get rid of the vanishing gradient.

$\Longrightarrow$ final result:

$$\frac{\partial h_{t+1}}{\partial h_t} = \text{diag}(z_t) + \text{diag}\left(\tanh'\left(W_{xh} x_{t+1} + r_{t+1} W_{hh} h_t + b_h\right)\right) \cdot \text{diag}(r_{t+1}) \cdot W_{hh}$$

↑ controls things.

We no longer study only W_{hh}, no more vanishing gradient.

Recording from Nov. 4th: (part 2)

Unsupervised Learning: no labels.

↳ Clustering: group things together based on similarity measure.

Assume k clusters $[C_1, \dots, C_k]$. All have some representative point $[z_1, \dots, z_k]$

$$\min_{j=1,\dots,k} \text{dist}(x_i, z_j)$$

cost function: $\sum_{j=1}^{k} \sum_{i \in C_j} \| x_i - z_j \|_2^2$

$$\text{dist}(x_i, z_j) = \| x_i - z_j \|_2^2$$

Two types: Hard, soft. ← varying degree of membership / confidence.

↳ $C_i \cap C_j = \{\phi\}$

Certainty

k-means clustering (Hard):

specify k num clusters.

for each cluster, randomly select $(z_1, \dots, z_k)$

$\min_{j=1,\dots,k} \|x_i - z_j\|_2^2$ for i size of dataset.

cost: $\sum_{i=1}^{n} \min_{j=1,\dots,k} \|x_i - z_j\|_2^2$

$\min_{z_1,\dots,z_k} \sum_{j=1}^{k} \sum_{i \in C_j} \|x_i - z_j\|_2^2 = \text{cost.}$

$\frac{\partial}{\partial z_j} \sum_{i \in C_j} \|x_i - z_j\|_2^2 = 2 \sum_{i \in C_j} x_i - z_j$

$\therefore \sum_{i \in C_j} x_i = \sum_{i \in C_j} z_j \Rightarrow z_j = \frac{\sum_{i \in C_j} x_i}{|C_j|}$

Only applies for Euclidean distance.

Alternating manner.

↳ Now, we update the representative point. Two separate steps for better computation.

Example:

Point A (2,3) $z_1 = (2,3)$

B (3,3) $z_2 = (6,5)$

C (6,5)

D (8,8)

Point A: $(2-2)^2 + (3-3)^2 = 0$ → z_1; $(2-6)^2 + (3-5)^2 = 20$ → z_2

$A \in C_1$

Point B: $(3-2)^2 + (3-3)^2 = 1$ → z_1; $(3-6)^2 \dots$ → z_2

$B \in C_1$

$C \in z_2$

$D \in z_2$

$z_1^{(2)} = \left(\frac{2+3}{2}, \frac{3+5}{2}\right) = (2.5, 4)$

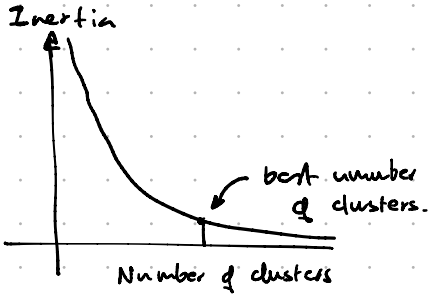
$z_2^{(2)} = (7, 6.5)$

} do calc. again

$A, B \in C_1^{(2)}$

$C, D \in C_2^{(2)}$

Done. stop the algo.



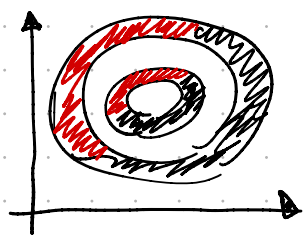
We can also use k-means for quantizing. k-means kernel clustering?

Initialization is important (Limitation)

↳ Interesting initialization perhaps. No manual clustering. k-means ++. Maximize distance to get better init.

Choose random rep. Then distance to all other points calculated. p(select next rep.) based on the dist. squared.

What if we have spherical clusters?



Not good clusters. ⟹ Kernel k-means clustering

Apply ϕ kernel function. Everything else is the same.

1. $\min_{j=1,\dots,k} \|\phi(x_i) - z_j\|_2^2$

2. $\min_{z_1,\dots,z_k} \sum_{j=1}^{k} \sum_{i \in C_j} \|\phi(x_i) - z_j\|_2^2$

$\Rightarrow z_j = \frac{\sum_{i \in C_j} \phi(x_j)}{|C_j|}$ Again, kernel trick: $\phi^T(x_j)\phi(x_i) = (1 + x_j^T x_i)^d$ for poly.

$\phi^T(x_j)\phi(x_i) = \exp(-\gamma \|x_j - x_i\|_2^2)$, for rbf.

$$\|\phi(x_i) - z_j\|_2^2 = \left\| \phi(x_i) - \frac{\sum_{i \in C_j} \phi(x_j)}{|C_j|} \right\|_2^2 = \phi^T(x_i)\phi(x_i) - 2\sum_{i \in C_j} \frac{\phi^T(x_i)\phi(x_i)}{|C_j|} + \sum_{t \in C_j} \frac{\phi^T(x_t)\phi(x_t)}{|C_j|^2}$$

k means only works for squared Euclidean distance. If another cost function, k-medoids.

1. Specify k
2. $(z_1, \dots, z_k) \subset (x_1, \dots, x_n)$
3. Iter.: $\sum_{i=1}^{n} \min_{j=1,\dots,k} dist(x_i, z_j)$

$\sum_{j=1}^{k} \sum_{i \in C_j} dist(x_i, z_j)$

for each z_j will go over $\{x_1, \dots, x_n\}$ & find based on lowest error.

→ not necessarily closed form, could be a search space

The rest is the same

Maybe cost function captures it better.

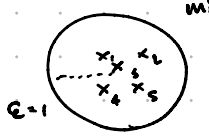
Data dependent problem.

Recording from Nov. 11th:

Density-based clustering: clusters are dense regions separated by low density regions.

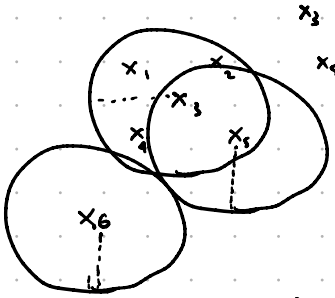
↳ DBScan: { core points: more than pts_{min}
border points: less than pts_{min}, in neighbourhood.
noise points: not core or border

Density: points within circle w/ radius r.



num. points in neighbourhood of x_2: 5 > 4.

x_2 core point.



x_3 core point

x_5 contains 3 < 4

∴ border point.

x_6 contains 1;

∴ noise point

This has nothing to do with centroids.

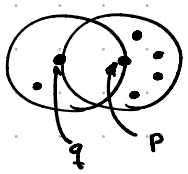
Only core points.

Algo. first finds all the core points, then checks for border & noise.

Check if x_j is in the neighbourhood of a core point, if points < pts_{min}.
If it is, border point. Else noise.

Directly density reachable: q ddr from p if p is a core point & $q \in \epsilon$-ball of p.

Assymetric



q ddr from p

p is not ddr from q

↳ because q is not a core point.

ε-ball of q < pts_{min}

There is also indirect density reachable.

DBscan algo.: ε & pts_{min}.

Identify core, border, noise points.

Choose core point randomly. All core points close to this assigned to same cluster, until all the core points assigned. ⟹ allows arbitrary shape.