4- convolutional neural networks

cnns = nn's that use convolution in place of general matrix mult at least one of ther layers
* suitable for tasks that has translation or time equivariance for data that has a grid topology
↳ examples = images (2d), videos (3d), time series (1d grid)

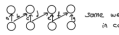
mlp vs cnn =
↳ mlp = fully connected, matrix multiplication is used to compute the next layer
↳ cnn = sparse connection, convolution is used to compute the next layer

convolution = computes the similarity of two signals
↳ in CNN's cross-correlation is used instead of convolution

connects
① sparse interaction = input layer $(4\times1)^{m}$ * filter $(2\times1)^{k}$ → output layer $(3\times1)^{n}$
↳ only 4 pixels are used in input layer to compute one output layer pixel
↳ complexity in fully connected layer = O(mn), in sparse = O(mk) (k is much less than n)
② parameter sharing = same kernel / filter is applied everywhere on a layer
↳ number of parameters to be learned and stored is dramatically reduced
③ equivariant representations = if we move object in the input, its features will move the same amount in the output
↳ convolution is commutative with translation $f(g(x)) = g(f(x))$
④ ability to process different input sizes = mlp only accepts a fixed-size input vectors

pooling = takes the output of the previous layer at a certain location L and computes a summary of the neighbor around L → introduces invariance to translation
↳ makes the model robust to the exact location of features → 8
↳ if pooling is performed with stride (step size) > 1, reduces computational complexity and memory requirements (reduces the size)

linear operations = matrix multiplication, convolution
non-linear operations = relu, sigmoid (activation functions), pooling

local connection = like convolution but no sharing  same weights are used in convolution

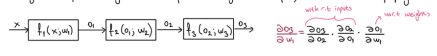
AlexNet = pioneering CNN known for winning the ImageNet (1000 classes) competition in 2012
↳ trained on 2 GPU (due to memory issues)
↳ 8 layers = 5 convolution + 3 full-connected
↳ convolution layer = convolution + relu + normalization (with respect to neighbors) + max pooling

reducing overfitting

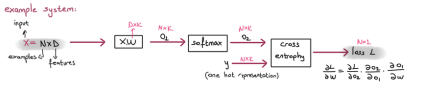
data augmentation (bigram) = artificially enlarging the dataset using label preserving transformations
↳ transformations, reflections, modifying the color channel intensities
dropout = set the output of each hidden neuron to with probability p (like 0.5) in training time
↳ for every input, the nn samples a different architecture
↳ reduces the dependence between neurons (memorizing)

implementation of backpropagation

modular approach = instead of deriving the gradient again and again, for each layer, derive the graints of each layer with respect to weights and input, use them in backpropagation



example system:



initialization of neural networks

① pre-training or transfer learning = use the weights of dataset trained with different dataset but still in the same domain, then continue training
② start from scratch = initialize with random numbers from normal distribution N(0,1)
↳ as the number of dimension increases, variance of the weights increases linearly
↳ w = random(n)/sqrt(n) or random(n)/sqrt(2/n) to prevent large responses (due to var)
↳ initializing with all zeros → vanishing gradients or very large responses
③ use the unsupervised pre-training method = not very common today

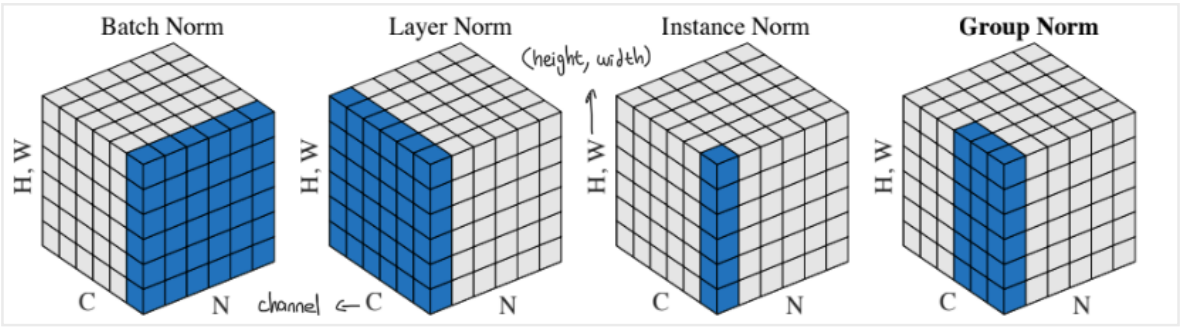
efficient implementation of a convolutional layer $\frac{200-(10-1)}{2}$

input = 200×200×3 every block of 10×10×3 → (300×1 vector) × 96×96 = 300×9216
conv layer = 75 filters 10×10×3, stride=2 → 75×300
W * X = (75×300) * (300×9216) = 75×9216 reshape back → 75×96×96 (number of channels)

normalization layers

batch normalization = maintains mean close to 0, and std dev close to 1
① take m instances
② calculate mean μ_B and var σ_B^2
③ normalise all $\hat{x}_i \leftarrow (x_i - \mu_B) / \sqrt{\sigma_B^2 + \epsilon}$
④ scale and shift $y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$
* γ and β are parameters to be learned
* take the avg γ and β, during test
* applied mostly after fully connected layer or before non-linearities
* make the model more robust to bad initialization
* regularize the model (preventing overfitting and generalizing better)

→ batch norm's batch size is more sensetive (error changes too much when batch size changes), while group norm's group size does not change the error much (not sensetive)

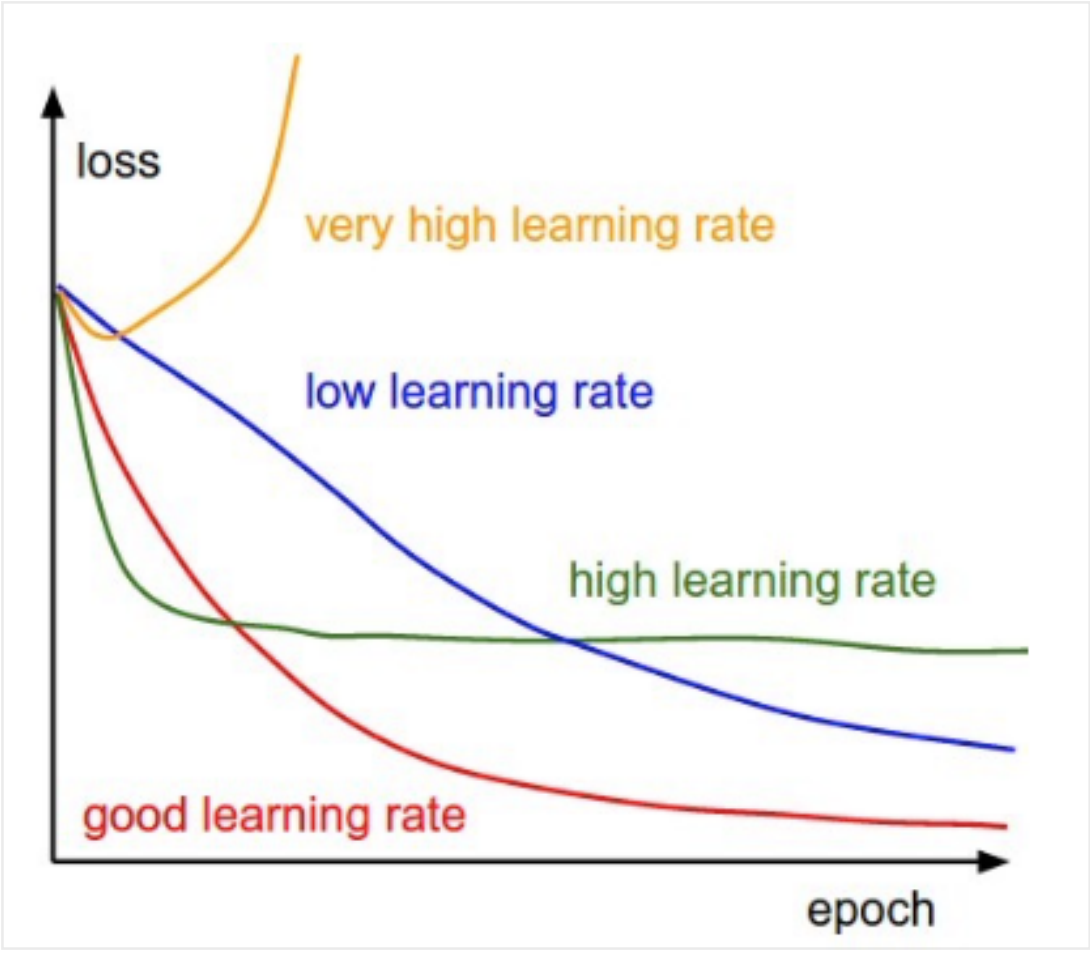


adaptive learning methods

- in stochastic gradient descent, decaying the learning rate is required, because the noise due to sampling m examples does not vanish even if it is local minimum
- if it is constant and low enough, it is guaranteed to make zero or positive converge

decaying learning rate:

- step decay (reduce by some factor after some epochs) → more practial
- exponential decay: $\alpha = \alpha_0 \cdot e^{-kt}$ (α_0, k are hyperparameters) * decreasing must be slow
- $1/t$ decay: $\alpha = \alpha_0/(1+kt)$



adaptive learning rate methods = use momentum to update learning rate

per parameter adaptive learning methods =

↳ delta-bar-delta = increase if partial derivative remains the sign, else decrease the l rate

↳ adagrad = decrease learning rate for weights with high gradients, increase small updated ones

↳ RMSprop = mini batch / moving avg of sum of squared gradients version of adagrad

↳ adam = RMSprop with momentum, uses smoothed gradient m (hyperparameter $\epsilon, \beta_1, \beta_2$)

image classification

imagenet large scale visual recognition challenge (ILSVRC) = 1.2 million images, 1000 categories

↳ task: given an image, among 5 predictions if one of them correct, it is success

↳ AlexNet (2012): 16.5% → ZF (2013): 11.7% → VGG (2014): 7.3% → GoogLeNet (2014): 6.7% →

Resnet: 3.6% → GoogLeNet-v4 (2016): 3.1% (error rates) ☆ human error rate = 5.1%

↳ residual network ↳ uses inception modules (1x1, 3x3, 5x5 conv and 3x3 max pooling)

plain network: x → layer → relu → layer → relu → H(x)

residual network: x → layer → relu → layer → ⊕ (H(x) = F(x) + x) → relu

identity x

DenseNet:  L(L+1) directions (densely connected) less # of parameters

object detection

☆ given an image and an object class, find its instance(s) on this image

convnet = generate huge number of object proposals / candidates → convnet → estimate the class

↳ done using convnet also

artistic style transfer = image → painting with given style

☆ convnet1 captures the style, convnet2 captures the content of input, convnet3 generates the output ↳ using textures

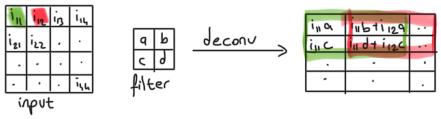
image segmentation = locating objects and boundries in images

upsampling = for imbalanced datasets, increasing the minority class

making learnable to deep learning libraries

↳ unsample + convolution

↳ deconvolution / transposed convolution



Grad-CAM = visually explains which part of the image is responsible for the class label

WaveNet = convnet used for speech / audio