neural networks: ConvNets, ResNets, batch normalization

Saeed Saremi

Assigned reading: 10, 9.5, 7.4

October 1, 2024

a summary of the previous lecture

- L-layer (feedforward) neural network as a certain type of nonlinear function:

$$a_j^{(\ell)} = \sum_{i \in [d_{\ell-1}]} \theta_{ji}^{(\ell)} x_i^{(\ell-1)}$$

$$x_j^{(\ell)} = g_\ell(a_j^{(\ell)}),$$

  where the computation is iterative from $\ell = 1$ to $\ell = L$; by convention $g_L(z) = z$.
- The neural network as a function is denoted by $f_\theta : \mathbb{R}^d \to \mathbb{R}^K$
- For *maximum likelihood* learning, the loss is always written as the ggreen sum of losses by individual points $i \in [n] := \{1, \ldots, n\}$ in the training set $\mathcal{D} = \{(x_i, y_i)\}_{i \in [n]}$: $\mathcal{L}(\theta) = \sum_{i=1}^n \mathcal{L}_i(\theta)$, where $\mathcal{L}_i(\theta)$ is short for:

$$\mathcal{L}_i(\theta) := \mathcal{L}(x_i, y_i; \theta) = \mathcal{L}(f_\theta(x_i), y_i).$$

- SGD (in its pure) form is defined by the following updates:

$$\theta_{t+1} = \theta_t - \epsilon_t \nabla_{\theta_t} \mathcal{L}_i(\theta_t),$$

  where $i \in [n]$ is selected at random at each iteration t, where $\theta \in \mathbb{R}^m$.
- What is the order of n, d, K, m?

backpropagation algorithm

We can compute the gradient $\nabla \mathcal{L}(\theta)$ in linear time $O(m)$:

$$\Delta_j^{(\ell)} := \partial_{a_j^{(\ell)}} \mathcal{L}$$

$$\partial_{\theta_{ji}^{(\ell)}} \mathcal{L} = \partial_{a_j^{(\ell)}} \mathcal{L} \; \partial_{\theta_{ji}^{(\ell)}} a_j^{(\ell)} = \Delta_j^{(\ell)} x_i^{(\ell-1)}$$

$$\Delta_j^{(\ell)} = \left(\sum_{k \in [d_{\ell+1}]} \Delta_k^{(\ell+1)} \theta_{kj}^{(\ell+1)} \right) g'(a_j^{(\ell)}),$$

where the errors $\Delta_j^{(\ell)}$ are backpropagated from $\ell = L$ to $\ell = 1$.

outline

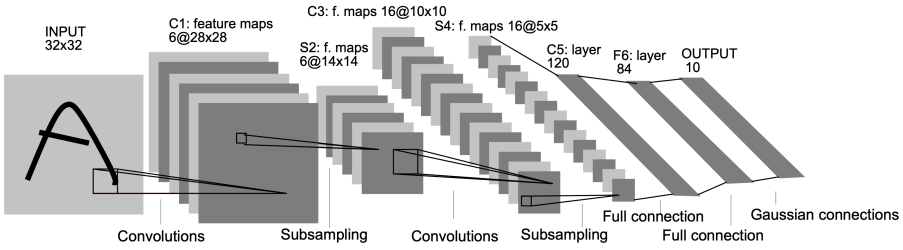


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Figure: *Gradient-based learning applied to document recognition, LeCun et al., 1998*

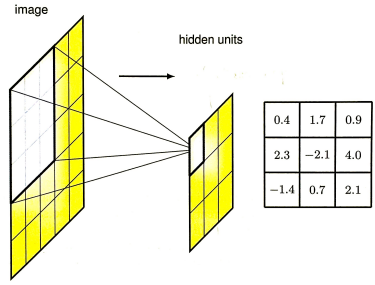
from feature engineering to neural architecture engineering

- fully connected networks are parameter hungry.
  - consider $d_0 = 10^6$ and a moderately large neural network with $d_1 = 10^3$.
  - in low-data regime, "too many parameters"[1] can lead to overfitting.
- for many problems of interests we have a preference, known as inductive bias, for certain types of solutions that the algorithm should learn.
  - One class of inductive bias goes under invariance/equivariance.
  - one can in principle learn these from data at a cost
  - why not bake our inductive biases into the neural architecture?
- In image recognition, ConvNets use a convolutional structure that applies filters across an entire image. This introduces an inductive bias: the assumption that objects in an image can appear anywhere, and patterns (like edges or textures) should be detected regardless of their location.

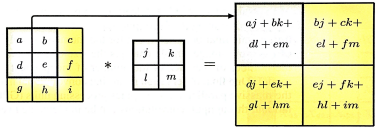
[1] take this with a grain of salt

invariance and equivariance

feature detectors (parameters as tiny images)

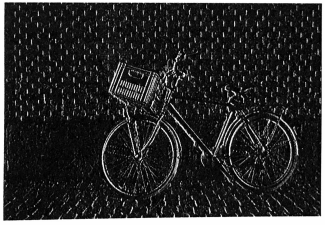


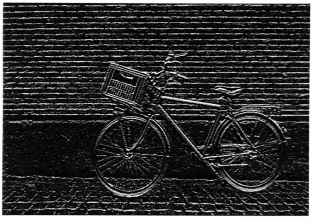
$$x^{(1)} = \mathsf{ReLU}(\theta^\top x^{(0)})$$



$$a^{(1)}(i_1, i_2) = \sum_{j_1 \in [F]} \sum_{j_2 \in [F]} x^{(0)}(i_1 + j_1, i_2 + j_2)\, \theta(j_1, j_2)$$

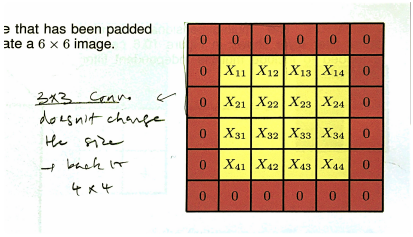
can you guess the 3×3 convolution filter **?**

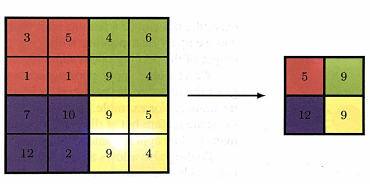




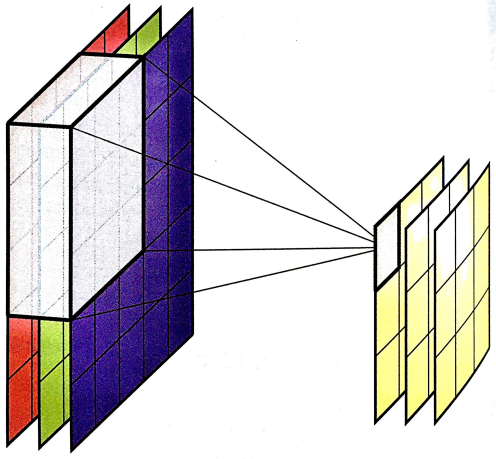
padding and strides

- ? What is the size of the feature map after convolving an image of size $W \times W$ with a kernel/filter of size $F \times F$?
- ? What is the size the feature map if we pad the image with P pixels?
- strided convolutions are used if $W \gg F$ to scale down the size of feature maps.
- pooling (typically *not learned*) has a similar effect but it also adds invariances





multidimensional convolution (C_{in})
and multiple feature maps (C_{out})



putting it all together

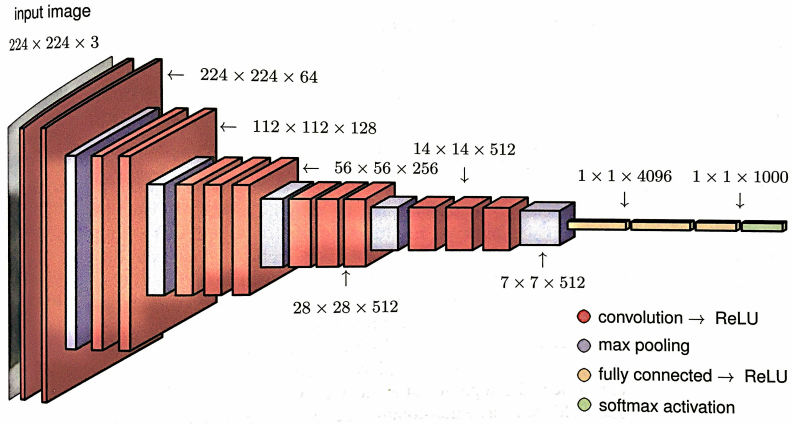
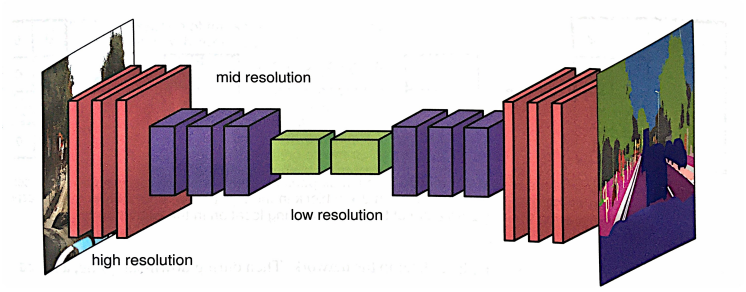


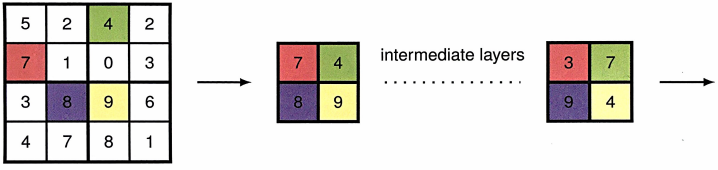
Figure: VGG-16 architecture for ImageNet classification

- AlexNet convolution filters 11×11, VGG-16 filters 3×3.
- 140 M parameters, 103 M of which is in the first fully connected layer

up-sampling (segmentation, denoising, etc)



- unpooling is an important step for up-sampling in ConvNets



visualizing trained ConvNets

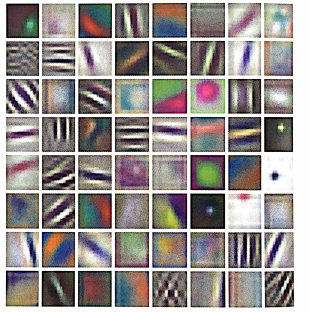


Figure: emergence of Gabor filters in the first layer of ConvNets

- the first layer is easy to visualize since the filters are images themselves!
- Gabor filters go back to 1946:

$$G(x_1, x_2) = \exp\left(-\frac{x_1^2 + x_2^2}{2\sigma^2}\right) \cos\Big(\omega\big(\cos(\theta)x_1 + \sin(\theta)x_2\big) + \phi\Big)$$

› Mathematica demonstration

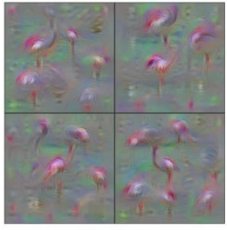
› Hubel and Wiesel demonstrations

poking neural nets using gradients: from $\nabla_\theta \mathcal{L}$ to $\nabla_x \mathcal{L}$

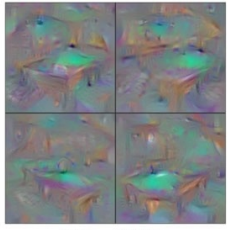
- so far we have used (stochastic) gradient descent for learning:

$$\theta_{t+1} = \theta_t - \epsilon_t \nabla_{\theta_t} \mathcal{L}_i(\theta_t).$$

- following learning we can use $\nabla_x a_i^{(\ell)}(x; \hat{\theta})$ to understand the neural net:



Flamingo



Billiard Table

School Bus

› this method is not principled (why?) and it needs some regularization...

adversarial examples

- one can also negatively optimize the loss

$$\underset{x \in \mathcal{N}_\epsilon(x)}{\operatorname{argmax}} \mathcal{L}(x, y; \hat{\theta})$$

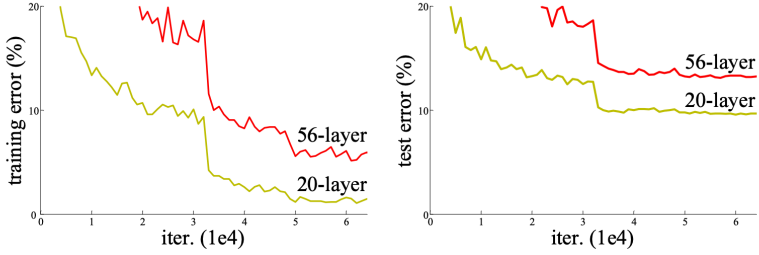
  and look for adversarial examples.

› a well-known algorithm is *fast gradient sign*:

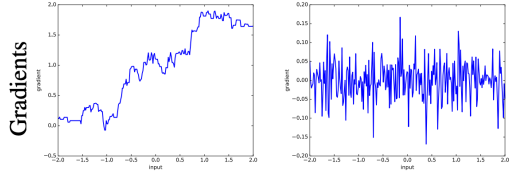
$$x \leftarrow x + \epsilon \operatorname{sign}\left(\nabla_x \mathcal{L}(x, y; \hat{\theta})\right)$$

? can you think of a way to make classifier robust to adversarial examples?

problems with training very deep nets



- one "surprising" empirical phenomenon in training the deep nets is the fact deeper is not always better
- this is related to the problem of vanishing gradient
- it is also related to the loss landscape becoming very rugged



ResNets

- One effective way to deal with this problem is to use skip connections

$$x^{(l+1)} = \mathcal{F}_{l+1}(x^{(l)}) + x^{(l)}$$

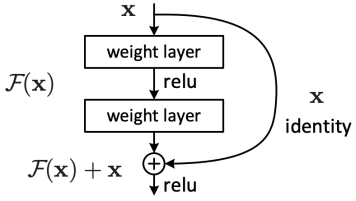
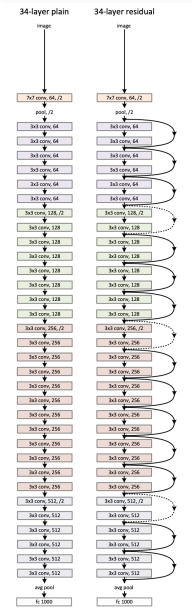


Figure 2. Residual learning: a building block.

Figure: *Deep Residual Learning for Image Recognition, He et al., 2015*



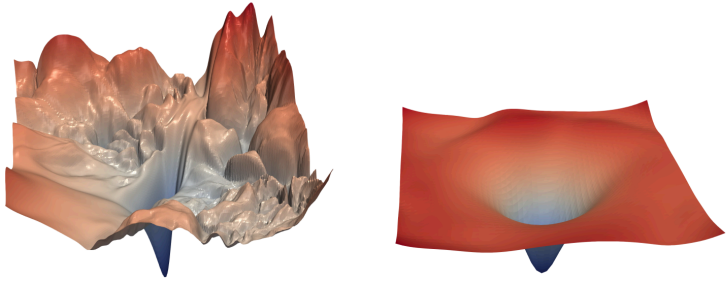
with ResNets, the deeper is better!

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

Figure: *Deep Residual Learning for Image Recognition, He et al., 2015*

the “loss lanscape” of resnets



(a) without skip connections (b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

Figure: *Visualizing the Loss Landscape of Neural Nets, Li et al., 2018*

good practices in training deep networks: normalization

- it is a good practice to normalize the INPUTs across different dimensions (why?):

$$\mu = \frac{1}{n}\sum_{i\in[n]} x_i,\ \sigma^2 = \frac{1}{n}\sum_{i\in[n]}(x_i-\mu)^2,\ x \leftarrow (x-\mu)/\sigma$$

- batch normalization: extend this normalization to activities in each hidden layer
  - these statistics change at each iteration of the SGD
  - these statistics should be gathered for a mini-batch of size b:

$$\mu^{(l)} = \frac{1}{n}\sum_{i\in[b]} a_i^{(l)},\ \sigma^{(l)} = \sqrt{\frac{1}{b}\sum_{i\in[b]}(a_i^{(l)}-\mu)^2},\ x^{(l)} \leftarrow \frac{x^{(l)}-\mu^{(l)}}{\sigma^{(l)}+\epsilon}$$

  - this is usually augmented by learnable parameters γ, β:

$$x^{(l)} \leftarrow \gamma x^{(l)} + \beta$$

  - the other big problem with BN is deciding on $(\mu^{(l)}, \sigma^{(l)})$ at inference time!
    - small batch training is bad (and multi GPU training is tricky)!
- alternative solution is layer normalization: in short change $i \in [b]$ to $i \in [d_l]$.