Learning by Descent

D.A. Forsyth

University of Illinois at Urbana Champaign

Learning the filters

- Procedure:
  - find many training pairs (noisy image, clean image)
  - adjust filters so that
  - Decode(Encode(noisy image)) is close to clean image
    - on average, over pairs
    - hope that this generalizes to new images
- Result:
  - Denoising autoencoder
  - Encoder has codes that represent images well
  - Opens the door to a lot of procedures

Find many training pairs

- No noise – system might ”cheat”
  - Produce a representation that isn’t useful

- What noise should you use?
  - Options:
    - Gaussian (but a fairly simple filter will deal with this)
    - Poisson (median filter)
    - knock out blocks of pixels (more challenging, and helpful)
    - blurring (ditto)
    - etc.

Adjusting the filters: notation

Write $\mathcal{E}(\cdot;\psi)$ for an encoder which accepts an image (in the $\cdot$ slot), produces an encoding, and has parameters ψ (the filter banks). Write $\mathcal{D}(\cdot;\phi)$ for a decoder that accepts an encoding ($\cdot$ slot again), produces an image, and has parameters ϕ (the filter banks). Stack the ψ and ϕ into one vector θ. Write $\mathcal{S}$ for a set of N training images. The i'th image is $\mathcal{I}_i$.

Adjusting the filters: loss

The autoencoder produces some image $\mathcal{O}(\mathcal{I}, \theta) = \mathcal{D}(\mathcal{E}(\mathcal{I}; \psi); \phi)$ when given $\mathcal{I}$. Construct a cost function $\mathcal{C}(\mathcal{O}(\mathcal{I}, \theta), \mathcal{I}_i)$ that compares the output of the autoencoder to $\mathcal{I}$. This cost function is typically a weighted combination of the L2 norm and the L1 norm (Section 9.2.2).

Now write

$$\mathcal{L}_{\mathcal{S}}(\theta) = \frac{1}{N} \sum_{i \in \mathcal{S}} \mathcal{C}(\mathcal{O}(\mathcal{I}_i, \theta), \mathcal{I}_i)$$

for the *loss* – an average over a set $\mathcal{S}$ of images of the cost per image. The problem is to find a θ that produce an acceptably small value of the loss. In an ideal world, $\mathcal{S}$ would be all possible images, but this isn't practical. Instead, train on some large set of images (the *training set*). If this set is large enough and representative enough, expect that the autoencoder will also have low loss on other images, a property called *generalization*.

Adjusting the filters: optimization problem, but weird

$$\mathcal{L}_{\mathcal{S}}(\theta) = \frac{1}{N} \sum_{i \in \mathcal{S}} \mathcal{C}(\mathcal{O}(\mathcal{I}_i, \theta), \mathcal{I}_i)$$

- Issues:
  - The cost function is very hard to evaluate (N is big)
  - There are lots of parameters (millions-billions)
    - so no newton's method
  - You don't actually want an optimum
    - you want a set of filters that works well on other images

Stochastic gradient descent

$$\mathcal{L}_{\mathcal{S}}(\theta) = \frac{1}{N} \sum_{i \in \mathcal{S}} \mathcal{C}(\mathcal{O}(\mathcal{I}_i, \theta), \mathcal{I}_i)$$

- Loss is a population mean
  - you can estimate this quite well with a sample mean
  - draw a small batch, average over that

Stochastic gradient descent

In the case of the loss function, choose a sample size B – usually called a *batch size* – draw $\mathcal{B}$, a set of B images $\mathcal{I}_j$ drawn uniformly and at random, and form

$$\nabla_\theta \mathcal{L}_{\mathcal{B}}(\theta) = \frac{1}{B} \sum_{j \in \mathcal{B}} \nabla_\theta \mathcal{C}(\mathcal{I}_j; \theta)$$

and use this as an estimate of

$$\nabla_\theta \mathcal{L}_{\mathcal{S}}$$

to take a descent step. Write

$$\hat{\nabla}_\theta \mathcal{L}$$

for this estimate. Choose a stepsize η_n for the n'th step, and the descent method becomes

$$\theta_{n+1} = \theta_n - \eta_n \hat{\nabla}_\theta \mathcal{L}.$$

This is *stochastic gradient descent* or *SGD*. Calling η_n a stepsize is dubious (the gradient isn't a unit vector); an alternative is to call it the *learning rate* (which isn't much better because it isn't a rate).

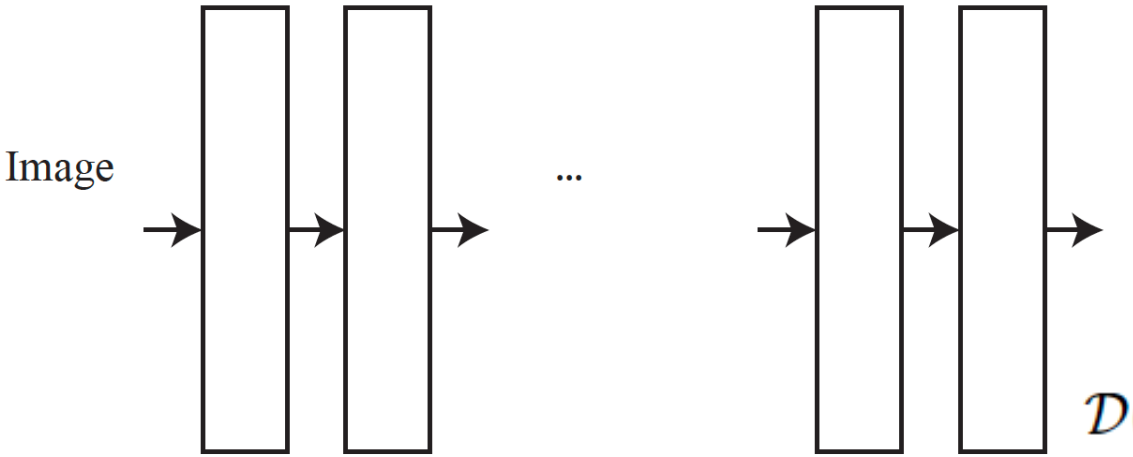
Stochastic gradient descent

$$\theta_{n+1} = \theta_n - \eta_n \hat{\nabla}_\theta \mathcal{L}.$$

- How big a step?
  - Line search
    - you can't – N is too big
  - Fixed length
    - too big (doesn't settle down)
    - too small (no progress)
  - Learning rate schedule
    - start biggish, take steps, make smaller
    - how big is biggish? try

Evaluating the gradient

$$\hat{\nabla}_\theta \mathcal{L} = \frac{1}{B} \sum_{j \in \mathcal{B}} \nabla_\theta \mathcal{C}(\mathcal{O}(\theta), \mathcal{I}_j).$$



$$\begin{aligned}
\mathcal{D}(\mathcal{E}(\mathcal{I};\psi);\phi) &= B_{k+1} \\
&\text{where} \\
B_{k+1} &= L_k(B_k;\theta_k) \\
B_k &= L_{k-1}(B_{k-1};\theta_{k-1}) \\
&\cdots \\
B_1 &= \mathcal{I}
\end{aligned}$$

Recursion from chain rule (Backpropagation)

$$
\begin{aligned}
\mathbf{u}_0^T &= \nabla_{\mathcal{O}} \mathcal{C}^T \\
\nabla_{\theta_k} \mathcal{C} &= \mathbf{u}_0^T \mathcal{J}_{L_k;\theta_k} \\
\mathbf{u}_1^T &= \mathbf{u}_0^T \mathcal{J}_{L_k;B_k} \\
\nabla_{\theta_{k-1}} \mathcal{C} &= \mathbf{u}_1^T \mathcal{J}_{L_{k-1};\theta_{k-1}} \\
&\dots \\
\mathbf{u}_r &= \mathbf{u}_{r-1}^T \mathcal{J}_{L_{k-r+1};B_{k-r+1}} \\
\nabla_{\theta_{k-r}} \mathcal{C} &= \mathbf{u}_r \mathcal{J}_{L_{k-r};\theta_{k-r}} \\
&\dots \\
\nabla_{\theta_1} \mathcal{C} &= \mathbf{u}_{k-1} \mathcal{J}_{L_1;\theta_1}
\end{aligned}
$$

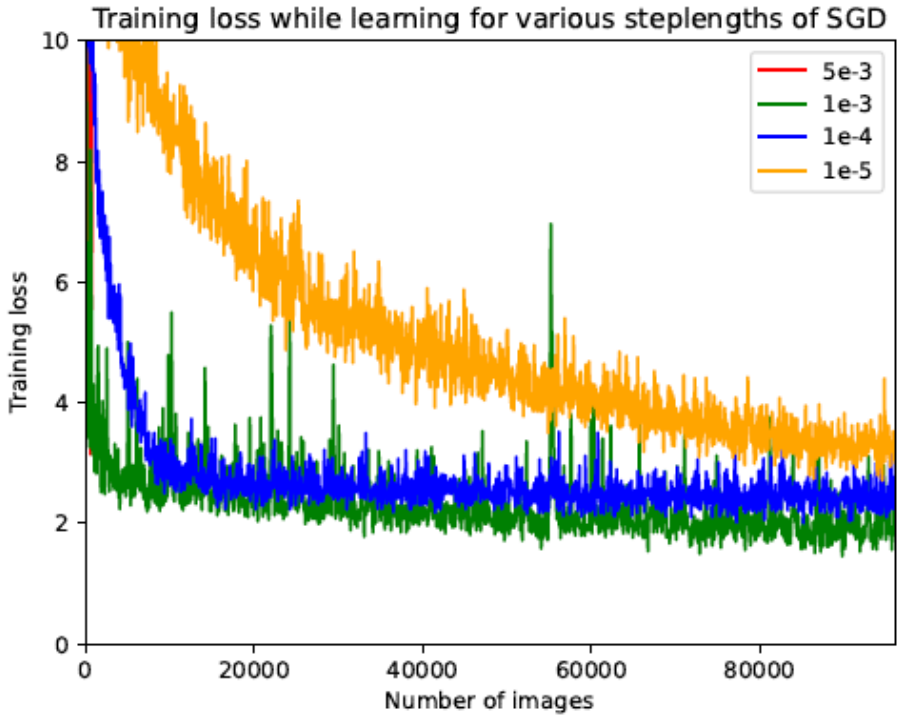
Derivatives of layer outputs with respect to parameters

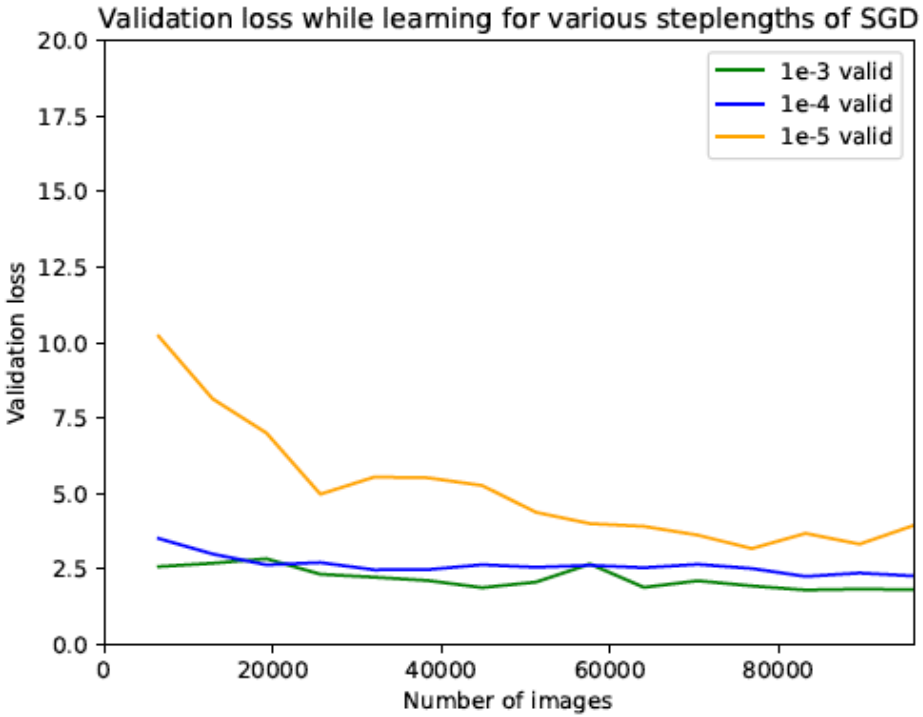
Derivatives of layer outputs with respect to inputs

Stochastic gradient descent

$$\theta_{n+1} = \theta_n - \eta_n \hat{\nabla}_\theta \mathcal{L}.$$

- Choice of steplength matters
  - more later

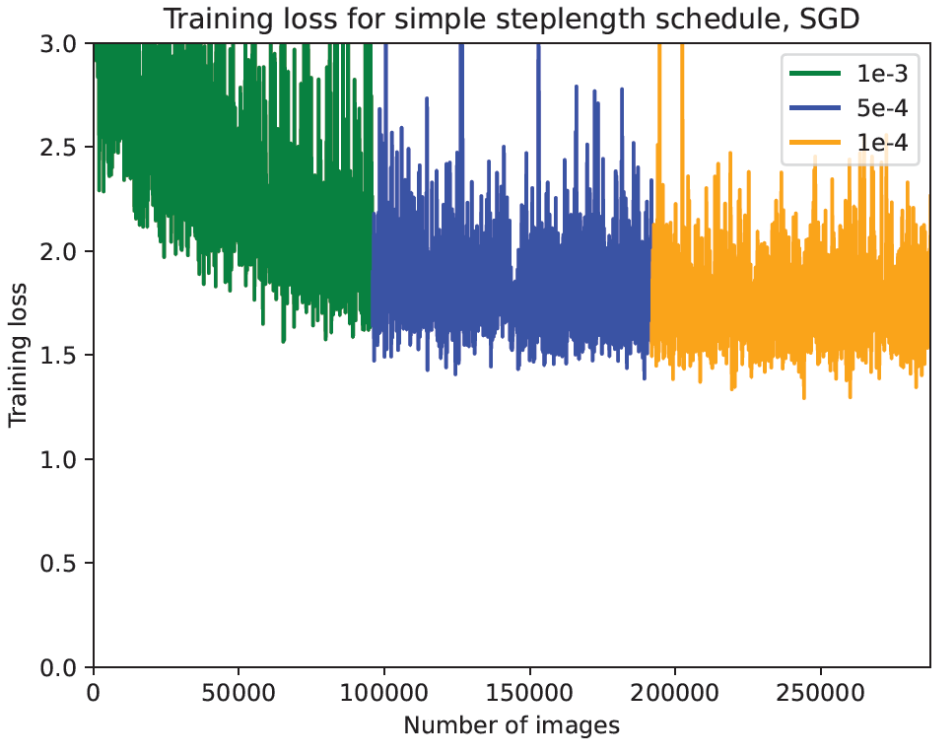


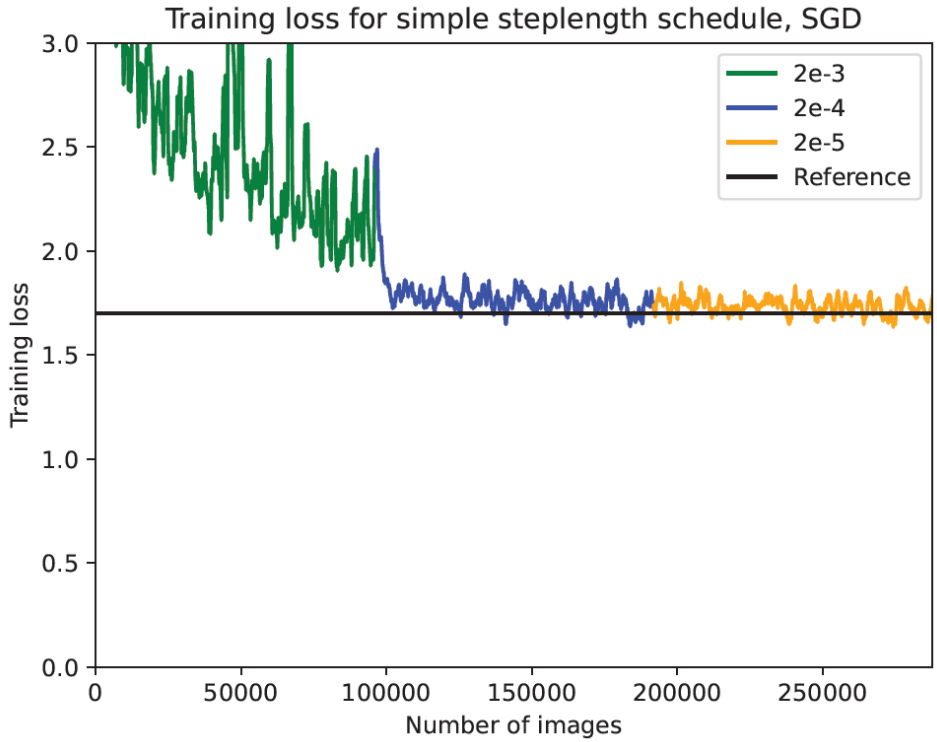


Stochastic gradient descent

$$\theta_{n+1} = \theta_n - \eta_n \hat{\nabla}_\theta \mathcal{L}.$$

- Steplength scheduling can help
  - more later





Things to think about...

17.6. Section 17.3.4 says: " Write $\mathcal{J}_{L_w;\theta_w}$ for the derivative of the function L_w with respect to parameters θ_w, and $\mathcal{J}_{L_w;B_w}$ for the derivative of the function L_w with respect to inputs B_w (recall these are matrices)". Why are these matrices? what are the elements of the matrices?