Project ideas

There will be a project report and a presentation due at the end of the semester.

To keep the number of presentations within reasonable limits, students are encouraged to form 2- or 3-person groups. The presentation and report should list the roles and contributions of each person in the group.

To assist you in getting started, there are a number of possible projects that you should be interested in. These projects are intended to involve some significant mathematical development, either in the creation or extension of an approach to machine learning in a particular problem, or in some analysis of an aspect of the system.

1. *Image registration (part 1)*: A gray-scale image can be thought of a function $g(x, y)$ over a rectangle $(x, y) \in [a, b] \times [c, d]$. We wish to register it with another (reference) image $h(u, v)$ through a transformation $\boldsymbol{\psi}\colon \mathbb{R}^2 \to \mathbb{R}^2$ so that $g(\boldsymbol{\psi}(\boldsymbol{u})) \approx h(\boldsymbol{u})$. The problem is to find $\boldsymbol{\psi}$ that best achieves this.

   This part is concerned with finding the best way of doing this: we want the function $\boldsymbol{\psi}$ to be smooth and invertible with both $\nabla\boldsymbol{\psi}$ and its inverse to be modest. Even if $\boldsymbol{\psi}$ is affine (that is, linear plus constant), this can be difficult to guarantee. Here is a representation of such a transformation that can be helpful:

$$\boldsymbol{\psi}(\boldsymbol{u};\boldsymbol{p}) = \left[\begin{array}{c|c} \cos\phi & -\sin\phi \\ \hline \sin\phi & \cos\phi \end{array}\right] \begin{bmatrix} \exp(\alpha) & \\ & \exp(\beta) \end{bmatrix} \left[\begin{array}{c|c} \cos\theta & \sin\theta \\ \hline -\sin\theta & \cos\theta \end{array}\right] \boldsymbol{u}+\boldsymbol{b}$$

   with parameter vector $\boldsymbol{p}$ containing the parameters α, β, θ, ϕ, and $\boldsymbol{b}$. Show that this $\boldsymbol{\psi}$ is invertible and differentiable. Minimizing

$$\int_R \left(g(\boldsymbol{\psi}(\boldsymbol{u};\boldsymbol{p})) - h(\boldsymbol{u})\right)^2 d\boldsymbol{u},$$

   where R is the rectangle on which $h(\boldsymbol{u})$ is defined, is a non-convex optimization problem. However, you can (and will) compute derivatives with respect to $\boldsymbol{p}$. You will need estimates of $\nabla g(\boldsymbol{x})$ which can be estimated from finite differences or other means. You should also look for limitations of this approach; for example, can reflections be represented in this way? Also, extend your method to include nonlinear transformations, but in a way that ensures that $\boldsymbol{\psi}$ is smooth and invertible. Can you explicitly find the inverse of the nonlinear $\boldsymbol{\psi}$ that you create?

2. *Image registration (part 2)*: Images typically have sharp boundaries which results in discontinuities and bad behavior for gradients. One way of handling this is to smooth out the image (represented as a function of two variables) using convolutions. The convolution of two functions (of two or more variables) is given by

$$(f * g)(\boldsymbol{y}) = \int_{\mathbb{R}^2} f(\boldsymbol{y} - \boldsymbol{x})\, g(\boldsymbol{x})\, d\boldsymbol{x}.$$

Approximate this by a sum, and implement this. To smooth out an image represented by f, take the convolution with the Gaussian function $p_\sigma(\boldsymbol{x}) = (2\pi)^{-1}\sigma^{-2}\exp(-\boldsymbol{x}^T\boldsymbol{x}/(2\sigma^2))$ where $\sigma > 0$ controls the amount of smoothing.

To apply this to the image registration task, we aim to minimize

$$\int_R \left((p_\sigma * g)(\boldsymbol{\psi}(\boldsymbol{u};\boldsymbol{p})) - (p_\sigma * h)(\boldsymbol{u})\right)^2 d\boldsymbol{u}.$$

What is $\nabla(p_\sigma * g)(\boldsymbol{x})$? How could this be computed in practice?

3. *Dimension reduction*. A common belief is that most realistic data sets can be approximated by a *manifold*, or higher dimensional curved surface, in an even higher dimensional space. Investigate this for the MNIST data set of hand-drawn digits.

A first cut at this is to perform an SVD on the data set. (You will first need to flatten the images into row or column vectors before performing the SVD.) How quickly do the singular values decrease?

But translating even a single image over a part of the plane will result in a very high-dimensional vector space of images, without giving significantly more information. If we have a translation vector $\boldsymbol{b}$, so that an image is shifted like this $h(\boldsymbol{x}) \mapsto h(\boldsymbol{x} - \boldsymbol{b})$, you can treat $\boldsymbol{b}$ as a two-parameter vector that helps to describe a given image, resulting in an extra two dimensions for the manifold. Translate all the MNIST images so that their center of mass (or brightness) is at the center of the image frame. Re-try the SVD on this set if centered images. How quickly do the singular values decrease now? Can other transformations be used to reduce the apparent dimension of the data set? What happens if you separate out a single digit (such as "9" or "3")? Would nonlinear transformations help represent these hand-drawn digits?

4. *Neural networks and optimization*. SGD is typically applied to neural networks in a very incomplete fashion. The task in this project is to see how close to a minimizer they really come. Gradients can be computed via backpropagation. But 2nd order derivatives are harder to get. Nevertheless, we can estimate 2nd derivatives in the form of matrix-vector products $\operatorname{Hess}\phi(\boldsymbol{w})\,\boldsymbol{d}$ of the Hessian matrix of ϕ at $\boldsymbol{w}$ with a vector $\boldsymbol{d}$ via finite differences:

$$\operatorname{Hess}\phi(\boldsymbol{w})\,\boldsymbol{d} \approx \frac{1}{2h}\left[\nabla\phi(\boldsymbol{w}+h\boldsymbol{d}) - \nabla\phi(\boldsymbol{w}-h\boldsymbol{d})\right] \qquad \text{for } h \approx 0.$$

   The eigenvalues (and eigenvectors) of $\operatorname{Hess}\phi(\boldsymbol{w})$ can be estimated using these matrix-vector products using various numerical algorithms such as the Lanczos and Arnoldi algorithms. A well known version of these is ARPACK. although originally written in Fortran, there are interfaces to ARPACK in various languages.

   Train a neural network for the MNIST data set, for example. (You can try this with other data sets, but this is easy to get and well-known.) Try to track how quickly $\|\nabla\phi(\boldsymbol{w}_k)\|$ reduces with the iteration number k (or number of epochs, if you wish). When "fully trained", use the above mentioned techniques to estimate the maximum and minimum eigenvalues of $\operatorname{Hess}\phi(\boldsymbol{w}^*)$. Does it seem to be close to a saddle point, or local minimizer?

5. *Particle swarm methods*. There are a large number of heuristic optimization methods based on biological analogies. These are perhaps better known as population-based optimization methods as they use a large number of "particles" to search for the minimizer of some objective function. However, there is typically very little theory to explain why they behave as they do, or to justify the results obtained.

   Pick one of these methods. These methods are typically stochastic with a small "learning rate". Approximate the motion by means of a system of stochastic differential equations. Can you derive an approximate equation for the "swarm" as a whole? That is, can you treat the "swarm" as a probability distribution that evolves in time? Are there barriers from using such an approach? Do the "particles" in the "swarm" attract each other an coalesce, or do they tend to mover away from each other. Note: there are a number of analytical tools that may be helpful, such as the Fokker–Planck equation for evolution of the probability distribution for the solution of a stochastic differential equation.

Even small steps in a task like this can be very helpful. Don't aim for complete theoretical understanding of these methods. Rather the focus should be on obtaining basic theoretical insights. Simulations can be helpful ***if they are focused on specific questions***.

6. *Extreme overfitting for a special kind of neural network.* There are papers claiming that certain kinds of neural networks have an interesting behavior as the number of hidden units goes to infinity. Specifically, for shallow neural networks (one hidden layer) with the ReLU activation function with *one* input (the input *vector* is just an input *number*) and one output, if the data set is fixed but the number of hidden units goes to infinity with randomly initialized weights and offset, then the limit function after training is a cubic spline interpolant of the data.

   Cubic splines are piecewise cubic polynomials (that is, on each interval $[x_i, x_{i+1}]$ the function is cubic, but the overall function is not only continuous but has continuous first and second derivatives). You should also review papers describing this behavior, and be willing to present a version of this theory. You should also try reproducing this behavior.

7. *The Google PageRank algorithm.* The foundation of Google's success is its web search algorithm. The algorithm uses only the network structure of web pages, and the ability to identify which web pages contain which search words. This algorithm is based on a Markov chain of a not-very-smart web surfer: at any moment the web-surfer is viewing a web page, and will go to another web page according to the following rule: the web surfer decides with probability $\alpha > 0$ to go to any of the available web pages with equal probability; otherwise, if the web page has links to other pages then one of them is chosen randomly with equal probability and the web surfer follows that link; if the web page has no outgoing links, then the surfer goes to any of the available web pages with equal probability.

   The ranking algorithm ranks web pages containing the chosen search words according to the equilibrium probability distribution of web pages for the Markov chain described in the previous paragraph. This requires the computation of the equilibrium distribution (or an approximation to it) for the above Markov chain. The matrix of web links is a very sparse matrix: there are billions of web pages, but each page has a relatively small number of links on average. This helps keep the cost of a single iteration of an algorithm like the Power Method relatively low. (Considering the number of

web pages, it will still be very expensive.) The use of $\alpha > 0$ mentioned above can make the method converge in a modest number of iteration. Describe this algorithm and how to implement it. Describe how quickly it converges, and why. You could try to apply it so some sources of information, such as the English Wikipedia pages and their links to each other.