CSC413
Lecture Notes

Yuchen Wang

February 14, 2020

Contents

1 Introduction & Linear Models 2

2 Multilayer Perceptrons & Backpropagation 2

3 Automatic Differentiation & Distributed Representations 2

4 Optimization 3

4.1 Bad critical points . . . 3

4.2 Ill-conditioned curvature . . . 4

4.2.1 Gradient descent dynamics . . . 4

4.2.2 Normalization . . . 5

4.3 Momentum . . . 6

4.4 Adam . . . 6

4.5 Stochastic Gradient Descent . . . 6

5 Convolutional Neural Networks & Image Classification 7

5.1 Different layers of CNN . . . 7

5.1.1 Convolution layer . . . 7

5.1.2 Pooling layers . . . 8

5.2 Size of a CNN . . . 8

5.3 Equivariance and Invariance . . . 8

5.4 Supervised Pre-training & Transfer Learning . . . 9

CS GRAD SCHOOL

ADMISSION CONSULTING

Do you want offers from top CS grad schools?

Yuchen Wang

MSCS@Stanford, SWE@Microsoft

www.yuchenwyc.com

Stanford University · UNIVERSITY OF TORONTO · Microsoft

My team offers

- Personal statements editing
- Resume editing
- Recommendation letter strategies
- Grad school/program list suggestion
- Application timeline planning
- 1:1 mentorship with top school Masters' and PhD graduates

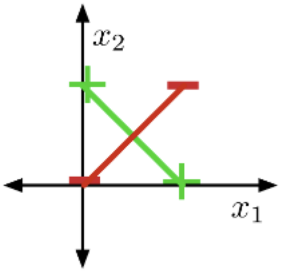
GET STARTED

✉ yuchenw@stanford.edu

1 Introduction & Linear Models

2 Multilayer Perceptrons & Backpropagation

Fact 2.1. XOR is not linearly separable.



Proof. Note that half-spaces are convex.
Suppose there is a plane that cut the input space into positive half-space and negative half-space.
Then the green line segment must be in the positive half-space, and the red line segment must be within the negative half-space.
But the intersection cannot lie in both half-spaces, which leads to a contradiction. ■

Definition 2.1 (feature)**.** A weight vector we learned in a neural network is called a feature.

3 Automatic Differentiation & Distributed Representations

Definition 3.1. Finite Differences One-side version:

$$\frac{\partial}{\partial x_i} f(x_1, \ldots, x_N) \approx \frac{f(x_1, \ldots, x_i + h, \ldots, x_N) - f(x_1, \ldots, x_i, \ldots, x_N)}{h}$$

Two-sided version:

$$\frac{\partial}{\partial x_i} f(x_1, \ldots, x_N) \approx \frac{f(x_1, \ldots, x_i + h, \ldots, x_N) - f(x_1, \ldots, x_i - h, \ldots, x_N)}{2h}$$

Remark 3.1. We often use finite differences to check our gradient calculations.

Autodiff v.s. finite differences

1. Autodiff computes exact gradients, whereas finite differences only computes an approximation.

2. Autodiff only requires a single forward pass and a single backward pass, and the backward pass is only a constant factor more expensive than the forward pass. Finite differences requires a separate forward pass for every entry of the gradient.

autograd.numpy.sum The functions in `autograd.numpy.sum` are reponsible for building the computation graph data structure to be used in backprop. The nodes in the graph are stored as `Node` instances, and the `Nodes` have attributes for the value, the function that was executed, and the arguments to the function (parents in the computation graph). The function `autograd.numpy.sum` retrieves the values from its `Node` arguments, feeds them to `numpy.sum`, and packages the result into a `Node` class.

4 Optimization

Problem	Diagnostics	Workarounds
incorrect gradients	finite differences	fix them, or use autodiff
local optima	(hard)	random restarts
symmetries	visualize **W**	initialize **W** randomly
slow progress	slow, linear training curve	increase α; momentum
instability	cost increases	decrease α
oscillations	fluctuations in training curve	decrease α; momentum
fluctuations	fluctuations in training curve	decay α; iterate averaging
dead/saturated units	activation histograms	initial scale of **W**; ReLU
ill-conditioning	(hard)	normalization; momentum; Adam; second-order opt.

4.1 Bad critical points

Definition 4.1 (saddle points)**.** A saddle point is a point where:

- $\nabla \mathcal{J}(\theta) = \mathbf{0}$
- $H(\theta)$ has some positive and some negative eigenvalues

Remark 4.1. If we're exactly on the saddle point, then we're stuck.

Definition 4.2 (plateaux)**.** A flat region is called a plateau.

Example 4.1. As follows:

- 0-1 loss
- hard threshold activations
- logistic activation with least squares

Definition 4.3 (permutation symmetry)**.** We can re-order the hidden units in a way that preserves the function computed by the network.

Fact 4.1. Training a multilayer perceptron is non-convex.

Proof. Assume for contradiction that there is a convex loss function for a multilayer perceptron.
Suppose we are given a layer which has parameters θ_1 and $|\theta_1| = K$.
By definition, if a function $\mathcal{J}$ is convex, then for any set of points $\theta_1, \ldots, \theta_N$ in its domain,

$$\mathcal{J}(\lambda_1\theta_1 + \ldots + \lambda_N\theta_N) \leq \lambda_1\mathcal{J}(\theta_1) + \ldots + \lambda_N\mathcal{J}(\theta_N)$$

for $\lambda_i \geq 0, \sum_i \lambda_i = 1$.
Because of permutation symmetry, there are $K!$ permutations of the hidden units such that

$$\mathcal{J}(\theta_i) = \mathcal{J}(\theta_1) \quad s.t. 2 \leq i \leq K!$$

If we take the average of the $N := K!$ permutations $\frac{1}{N}\sum_{i=1}^{N}\theta_i$, we get a degenerate network where all the hidden units are identical, which should have a higher cost.
However, by convexity we have

$$\begin{aligned}\mathcal{J}(\frac{1}{N}\sum_{i=1}^{N}\theta_i) &\leq \sum_{i=1}^{N}\frac{1}{N}J(\theta_i)\\ &= \frac{1}{N}\sum_{i=1}^{N}\mathcal{J}(\theta_i)\\ &= J(\theta_1)\end{aligned}$$

Meaning that this solution would have to be better than θ_1 for any arbitrary choice of θ_1, which leads to a contradiction.

■

Definition 4.4 (saturated unit)**.** A unit whose input z_i is in the flat region of its activation function.

Definition 4.5 (dead unit)**.** If there is a ReLU unit whose input z_i is always negative, the weight derivatives will be exactly 0. We call this a dead unit.

4.2 Ill-conditioned curvature

Definition 4.6 (ill-conditioned curvature)**.** Suppose the Hessian matrix H has some large positive eigenvalues (i.e. high-curvature directions) and some eigenvalues close to 0 (i.e. low-curvature directions), then gradient descent would bounce back and forth in high curvature directions and make slow progress in low curvature directions. (the gradient is perpendicular to the contours). This is known as ill-conditioned curvature, which is very common in neural network training.

4.2.1 Gradient descent dynamics

Consider a convex quadratic objective

$$\mathcal{J}(\theta) = \frac{1}{2}\theta^T A\theta$$

where A is a PSD symmetric matrix.
Then we would update θ using gradient descent:

$$\begin{aligned}
\theta_{k+1} &\leftarrow \theta_k - \alpha \nabla \mathcal{J}(\theta_k) \\
&= \theta_k - \alpha A \theta_k \\
&= (I - \alpha A)\theta_k \\
\implies \theta_k &= (I - \alpha A)^k \theta_0 \\
&= (I - \alpha Q \Lambda Q^T)^k \theta_0 && \text{(by Spectral Decomposition)} \\
&= [Q(I - \alpha \Lambda) Q^T]^k \theta_0 \\
&= Q(I - \alpha \Lambda)^k Q^T \theta_0 && (\text{Q orthogonal} \implies Q^T Q = I)
\end{aligned}$$

Hence, in the Q basis, each coordinate gets multiplied by $(1 - \alpha\lambda_i)^k$ where the λ_i are the eigenvalues of A.
Cases:

- $0 < \alpha\lambda_i \leq 1$: decays to 0 at a rate that depends on $\alpha\lambda_i$
- $1 < \alpha\lambda_i \leq 2$: oscillates
- $\alpha\lambda_i > 2$: unstable (diverges)

(the proof is omitted here)
Hence we need to set $\alpha < \frac{2}{\lambda_{max}}$, and this bounds the rate of progress by

$$\alpha\lambda_i < \frac{2\lambda_i}{\lambda_{max}}$$

Generalization The toy problem

$$\mathcal{J}(\theta) = \frac{1}{2}\theta^T A \theta$$

can be easily generalized to a quadratic not centered at zero

$$\mathcal{J}(\theta) = \frac{1}{2}\theta^T A \theta + b^T \theta + c$$

since the gradient descent dynamics are invariant to translation.

Since a twice-differentiable cost function is well approximated by a convex quadratic (though second-order Taylor approximation) in the vicinity of a (local) optimum, this analysis is a good description of the behaviour of gradient descent near a (local) optimum.
Therefore, in the case of ill-conditioned curvature, the convergence rate is slower since $\frac{2\lambda_i}{\lambda_{max}}$ is small for some of the λ_is, so that it makes slow progress towards the optimum.

4.2.2 Normalization

To avoid these problems, it's a good idea to center your inputs to zero mean and unit variance.

$$\tilde{x}_j = \frac{x_j - \mu_j}{\sigma_j}$$

Hidden units may have non-centered activations, in this case, we can replace logistic units (which range from 0 to 1) with tanh units (which range from -1 to 1).

4.3 Momentum

Even with normalization tricks, we still need algorithms to better deal with ill-conditioned curvature.

Definition 4.7 (Momentum)**.**

$$\mathbf{p} \leftarrow \mu\mathbf{p} - \alpha\frac{\partial \mathcal{J}}{\partial \theta}$$
$$\theta \leftarrow \theta + \mathbf{p}$$

where

- α is the learning rate, just like in gradient descent
- μ is the damping parameter. It should be slightly less than 1 (e.g. 0.9 or 0.99), or otherwise the ancient history would not decay and would affect the new gradient as much as the latest ones.

The effects:

- In the high curvature directions, the gradients cancel each other out, so momentum dampens the oscillations.
- In the low curvature directions, the gradients point in the same direction, allowing the parameters to pick up speed.

Remark 4.2. If the gradient is constant (i.e. the cost surface is a plane), the parameters will reach a terminal velocity of

$$-\frac{\alpha}{1-\mu} \cdot \frac{\partial \mathcal{J}}{\partial \theta}$$

This suggests if you increase μ, you should lower α to compensate.

4.4 Adam

Even with momentum and normalization tricks, narrow ravines are still one of the biggest obstacles in optimizing neural networks.
There is an optimization procedure called Adam which uses just a little bit of curvature information and often works much better than gradient descent.

4.5 Stochastic Gradient Descent

Definition 4.8 (batch training)**.** Computing the gradient requires summing over all of the training examples. This is known as batch training:

$$\nabla\mathcal{J}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \nabla J^{(i)}(\theta)$$

Remark 4.3. Batch training is impractical if you have a large dataset (e.g. millions of training examples)

Definition 4.9 (mini-batch)**.** Compute the gradients on a medium-sized set of training examples, called a mini-batch.

Definition 4.10 (epoch)**.** Each entire pass over the dataset is called an epoch.

5 Convolutional Neural Networks & Image Classification

5.1 Different layers of CNN

Each layer consists of several feature maps / channels, each of which is an array. If the input layer represents a grayscale image, it consists of one channel. If it represents a color image, it consists of three channels.

5.1.1 Convolution layer

Definition 5.1 (convolution - 1D)**.** If a and b are two arrays,

$$(a * b)_t = \sum_\tau a_\tau b_{t-\tau}$$

Definition 5.2 (convolution - 2D)**.** If A and B are two $2D$ arrays, then:

$$(A * B)_{ij} = \sum_s \sum_t A_{st} B_{i-s,j-t}$$

Property 5.1. Some properties of convolution:

1. Commutativity
$$a * b = b * a$$

2. Linearity
$$a * (\lambda_1 b + \lambda_2 c) = \lambda_1 a * b + \lambda_2 a * c$$

Definition 5.3 (convolutional layers)**.** The convolution layer has a set of filters. Its output is a set of feature maps, each one obtained by convolving the image with a filter.
It's common to apply a linear rectification nonlinearity:

$$y_i = \max(z_i, 0)$$

Remark 5.1. Why do we need activation functions?

- Convolution is a linear operation. Therefore, we need a nonlinearity, otherwise 2 convolution layers would be no more powerful than 1.
- If not applying activation functions, then two edges in opposite directions may cancel each other out.

5.1.2 Pooling layers

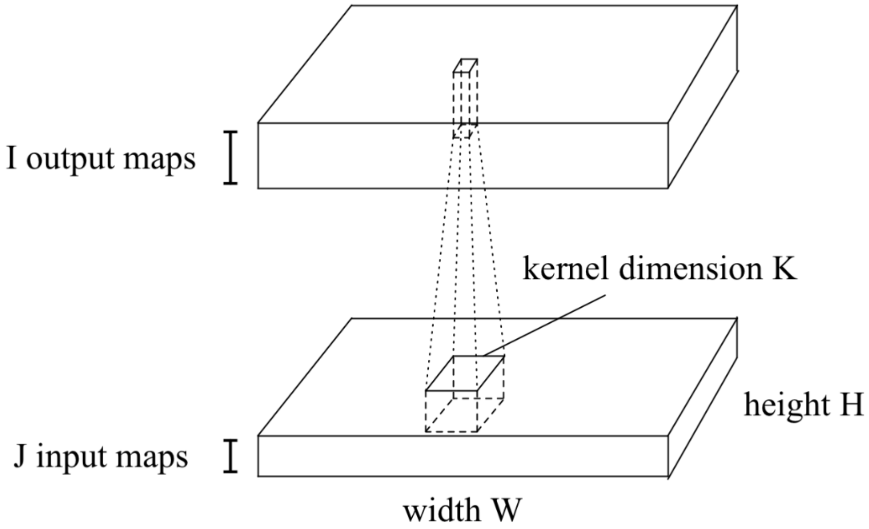
Definition 5.4 (pooling layers)**.** Pooling layers reduce the size of the representation and build in in variance to small transformations.

Definition 5.5 (max-pooling)**.** Max-pooling computes the maximum value of the units in a pooling group:

$$y_i = \max_{j \text{ in pooling group}} z_j$$

Remark 5.2. Because of pooling, higher-layer filters can cover a larger region of the input than equal-sized filters in the lower layers.

5.2 Size of a CNN



	fc layer	conv layer
# output units	WHI	WHI
# weights	W^2H^2IJ	K^2IJ
# connections	W^2H^2IJ	WHK^2IJ

Remark 5.3. Rules of thumb:

- Most of the units and connections are in the convolution layers.
- Most of the weights are in the fully connected layers.

5.3 Equivariance and Invariance

Definition 5.6 (equivariant)**.** A layer is equivariant to translation if you translate the inputs, the outputs are translated by the same amount.

Example 5.1. Convolution layers are equivariant.

Definition 5.7 (invariant)**.** A layer is invariant to translation if you translate the inputs, the prediction should not change.

Example 5.2. Pooling layers provide invariance to small translations.

5.4 Supervised Pre-training & Transfer Learning

I have no idea why raina didnhbgtgg’t read documentation gh