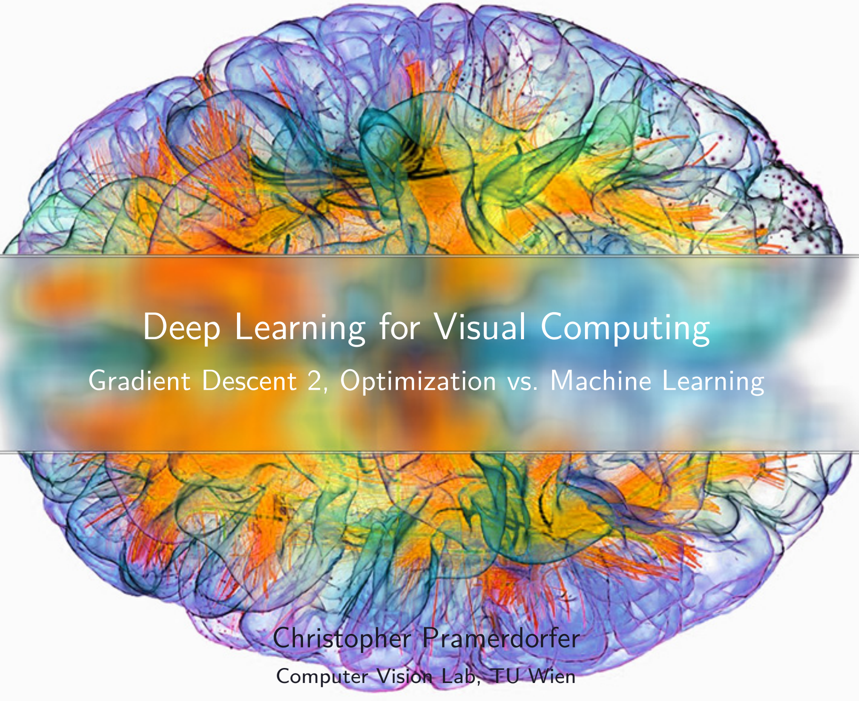


Deep Learning for Visual Computing

Gradient Descent 2, Optimization vs. Machine Learning

Christopher Pramerdorfer
Computer Vision Lab, TU Wien

Image by Neuroscape Lab

Topics

Minibatch gradient descent

Parameter initialization

Image preprocessing

Optimization vs. Machine Learning

Regularization

Optimization

We now have everything for training any parametric model

- A loss function (cross-entropy loss)
- An optimization algorithm (gradient descent)

Optimization

Open questions before we can start

- How to select $\mathcal{D}$ (data for loss function)
- How to initialize parameters $\boldsymbol{\theta}$ properly
- How to actually compute gradient (later)

Selecting $\mathcal{D}$

Recall that loss (and thus gradient) computed over data $\mathcal{D}$

- $\mathcal{D} = \{(\mathbf{x}_s, \mathbf{w}_s)\}_{s=1}^{S}$ can be any subset of training data
- Loss composed of S per-sample losses

$$L(\boldsymbol{\theta}) = \frac{1}{S} \sum_{s=1}^{S} H(\mathbf{w}_s, \mathrm{softmax}(f(\mathbf{x}_s; \boldsymbol{\theta})))$$

Selecting $\mathcal{D}$
Batch Gradient Descent

$\mathcal{D}$ can be chosen freely (subset of training set)

Obvious choice : use whole training set

- Want to use all data we have

Resulting algorithm called batch gradient descent

Selecting $\mathcal{D}$

Batch Gradient Descent

With gradient descent

- Evaluating gradient scales linearly with S
- Each iteration requires one gradient evaluation
- Many iterations required for convergence

With batch gradient descent

- Evaluating gradient requires run through whole training set
- One such run is called epoch

Selecting $\mathcal{D}$
Batch Gradient Descent

Number of iterations equals number of epochs

- Does not scale to large datasets

But complex models require large datasets

- DL datasets can have millions of samples

Selecting $\mathcal{D}$

Minibatch Gradient Descent

Overcome this problem by

- Processing the whole training set
- In minibatches of size S (one per iteration)

Possible because gradient is an expectation

- Can estimate training set loss on subset
- Also applies for the gradient

Selecting $\mathcal{D}$
Minibatch Gradient Descent

Resulting algorithm called minibatch gradient descent

- With $S = 1$ called Stochastic Gradient Descent (SGD)
- In practice often called SGD even if $S > 1$

Time for single gradient evaluation now constant

- Independent of dataset size

Selecting $\mathcal{D}$
Minibatch Gradient Descent

S varies between 1 and a few hundred samples

- Most common are 64, 128, 256
- 2^n for efficiency (data parallelism)

Decreasing S also decreases

- Computation time per iteration
- Memory required on GPU (minibatch processed as whole)
- Accuracy of the gradient estimate

Selecting $\mathcal{D}$
Minibatch Gradient Descent

Decreasing S causes more noisy gradient estimates

- Detrimental for optimization
- But can actually improve generalization performance

Increasing S

- Increases accuracy of gradient estimate
- With with less than linear returns

Selecting $\mathcal{D}$
Minibatch Gradient Descent

Important to sample minibatches randomly

- To break (possible) ordering in dataset

Standard approach in practice

- Shuffle training set once or before every epoch
- Process sequentially in minibatches

blank page

Gradient Descent in Action

Classification

Classification with $D = 2$ and $T = 2$

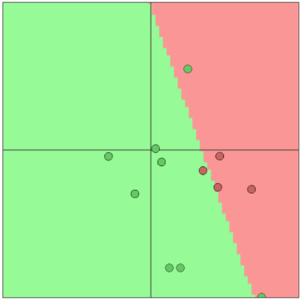
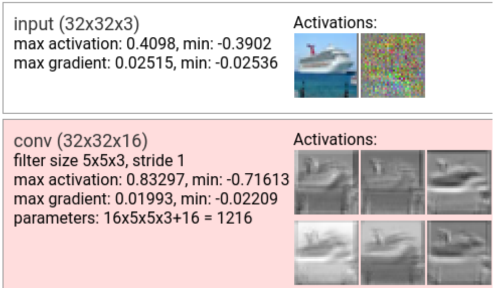


Image from cs.stanford.edu

Gradient Descent in Action

Image Classification

Classification on CIFAR10 using a CNN


input (32x32x3)
max activation: 0.4098, min: -0.3902
max gradient: 0.02515, min: -0.02536

Activations:

conv (32x32x16)
filter size 5x5x3, stride 1
max activation: 0.83297, min: -0.71613
max gradient: 0.01993, min: -0.02209
parameters: 16x5x5x3+16 = 1216

Activations:

Image from cs.stanford.edu

Gradient Descent in Action
Regression

Replace cross-entropy loss by squared loss

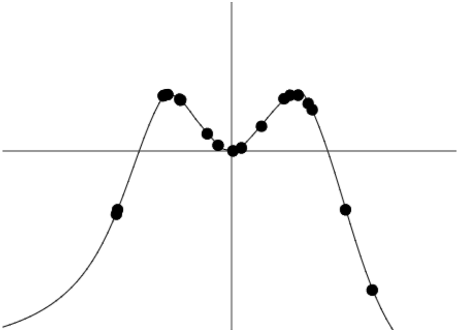


Image from cs.stanford.edu

blank page

Parameter Initialization

Linear models $\mathbf{w} = \mathbf{W}\mathbf{x} + \mathbf{b}$ have two types of parameters

- Multiplicative weights $\mathbf{W}$
- Additive biases $\mathbf{b}$

And so do CNNs, so following applies to them too

For simplicity consider single row $\mathbf{a}$ of $\mathbf{W}$, so $w = \mathbf{a}\mathbf{x} + b$

- We call w activation

Parameter Initialization
Biases

Less critical due to additive influence on activation

Usually initialized to zero

Parameter Initialization
Weights

Critical due to multiplicative influence on activation

Good initialization is important

- Determines whether gradient descent can converge
- Affects convergence rate and quality of result

Parameter Initialization
Weights

Rows of weight matrix must differ to break symmetry

- If not, gradient descent will apply the same updates
- And rows will remain identical (details later)

Initialize weights randomly from $\gamma\,\text{Norm}(0,1)$

- $\text{Norm}(0,1)$ is standard normal distribution

Parameter Initialization
Weights

Hyperparameter γ affects magnitude of weights

- Variance of $\gamma\,\text{Norm}(0,1)$ is γ^2

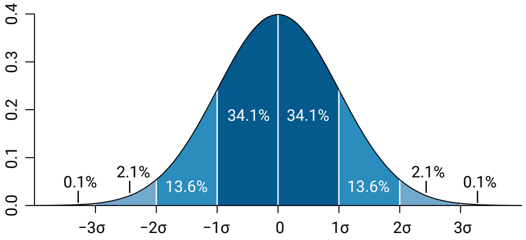


Image from wikipedia.org

Parameter Initialization
Weights

Hyperparameter γ affects magnitude of weights

- Thus that of activation
- Consequently magnitude of gradient and step size

If too large, activation (gradient, step size) explodes

If too small, activation vanishes and gradient descent stalls

Small weights usually preferred for regularization (details later)

Parameter Initialization
Weights

Magnitude of activation also depends on $\mathbf{x} = (x_1, \ldots, x_D)$

- We assume normalized data
- x_d have zero mean, unit variance (details later)

Parameter Initialization
Weights

Good heuristic for selecting γ : preserve variance of input

Let $y = \mathbf{ax} = a_1 x_1 + \cdots + a_D x_D$

We have $\mathrm{Var}(a_d x_d) = \mathrm{Var}(a_d)\,\mathrm{Var}(x_d) = \gamma^2$

- Weighs and inputs have zero mean
- Weights have variance γ^2, inputs have unit variance

Parameter Initialization
Weights

It follows that $\mathrm{Var}(y) = D\gamma^2$

- Variances add (assuming a_d and x_d are independent)

Variance of output is variance of input times $D\gamma^2$

- In our case input variance is 1

To preserve input variance we require $D\gamma^2 = 1$

- Thus we should set $\gamma = 1/\sqrt{D}$

Parameter Initialization
Weights

This is known as (a version of) Xavier initialization

- Popular choice in DL for linear layers

In DL

- Popular choice for linear layers
- Similar to best initialization for convolutional layers
- Details later

blank page

Preprocessing

Common to preprocess image data

Two reasons

- Gain invariance to undesired variations (per sample)
- Support training (over training set)

Preprocessing
Brightness and Contrast Normalization

Usually average image brightness should not affect result

- No effect on image content and thus class
- Achieved via mean subtraction

Usually also applies for average contrast

- Divide by standard deviation after mean subtraction
- Or use e.g. histogram equalization

Preprocessing
Brightness and Contrast Normalization

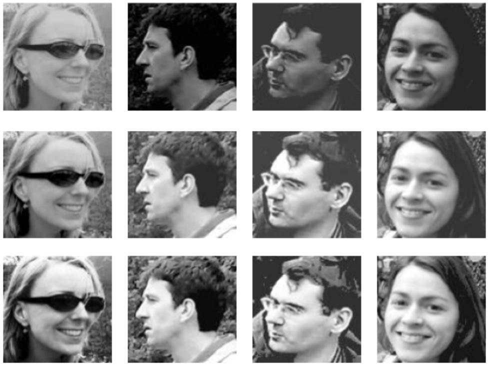


Image adapted from [1].

Preprocessing
Normalization

Transform training set to have zero mean, unit variance

- Compute mean and standard deviation
- Then divide each sample by standard deviation over all

For color images done for each channel

Normalized data is assumed

- By default parameters of algorithms (e.g. learning rate)
- By parameter initialization strategies

Preprocessing
Normalization

Popular variant is to normalize per feature

- Independently for each pixel/channel

Not required unless assumed that pixels are scaled differently

- Usually not the case for naturalistic images
- But still often done in practice

Preprocessing

Statistics must be computed from training set only

Same operations must be applied to validation and test sets

Optimization vs. Machine Learning

Looked at training from pure optimization perspective

- Find parameters $\hat{\theta}$ that minimize training loss
- Known as empirical risk minimization

Prone to overfitting

- Training data must capture underlying distribution well
- Almost never the case in image analysis

Optimization vs. Machine Learning

In machine (and deep) learning primary goal is different

- Obtain model that performs well on unseen data
- One with low test error

So we

- We minimize the training loss
- Hoping this also minimizes the test error
- Recall that this is possible due to correlated data

Regularization

The purpose of regularization is to

- Reduce the test error (e.g. increase accuracy)
- At the possible expense of training error/loss

Most strategies improve generalization

- By decreasing the model variance (increasing bias)
- Thus combating overfitting

Regularization
Weight Decay

Weight decay (L2 weight regularization) is very common

Penalizes large weights (not biases)

- Preventing certain inputs from dominating activation
- Thus encourages model to use all inputs

Weights must be initialized to have zero mean

Regularization
Weight Decay

Implemented by adding regularization term to loss function

$$L_{\text{reg}}(\boldsymbol{\theta}) = \frac{\delta}{2}\|\mathbf{w}\|^2 + L(\boldsymbol{\theta})$$

$\mathbf{w} \subset \boldsymbol{\theta}$ is vector of all weights

- Global regularization strength δ
- Can also differ on a per-weight (or per-layer in DL) basis

Regularization
Weight Decay

Ignoring bias, resulting gradient is $\nabla L_{\text{reg}}(\mathbf{w}) = \delta\mathbf{w} + \nabla L(\mathbf{w})$

- $\|\mathbf{w}\|^2 = \mathbf{w}^\top\mathbf{w}$, so gradient is $2\mathbf{w}$ (product rule)

So gradient descent update becomes

$$\begin{aligned}\mathbf{w} &= \mathbf{w} - \alpha(\delta\mathbf{w} + \nabla L(\mathbf{w})) \\ &= \mathbf{w} - \alpha\delta\mathbf{w} - \alpha\nabla L(\mathbf{w}) \\ &= (1 - \alpha\delta)\mathbf{w} - \alpha\nabla L(\mathbf{w})\end{aligned}$$

Regularization
Weight Decay

Weights are shrinked by constant factor before each update

- Thus weights decay to zero (hence the name)

Decay strength δ is another hyperparameter

- No effect if too small
- Dominates data loss (e.g. cross-entropy) if too large

Regularization
Early Stopping

When training models capable enough to overfit (e.g. CNNs)

- Training error decreases steadily
- But validation error begins to rise again a some point

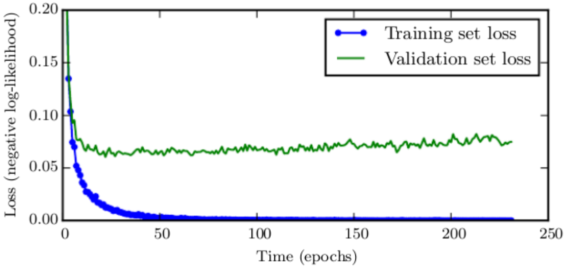


Image from [2]

Regularization

Early Stopping

We obtain model with lower validation error

- And thus (hopefully) lower test error
- By returning to parameters with lowest validation error

Implementation

- Maintain copy of weights with lowest validation error
- Stop training when validation error does not improve
- For a manually specified number of iterations/epochs

Regularization
Early Stopping

Most common form of regularization in DL

Works together with other strategies (e.g. weight decay)

Easy to implement

Requires a validation set

- Which we have anyways for hyperparmeter optimization

Regularization

In DL we virtually always use early stopping and weight decay

In addition to other regularization strategies

Because DL models work best

- If they have enough capacity to overfit
- But are regularized properly

Bibliography

[1] Prince, *Computer Vision Models*. 2012.

[2] *Deep learning*, 2016, [Online]. Available: http://www.deeplearningbook.org.