Regularization and Optimization

Steve Avsec

Illinois Institute of Technology

March 18, 2024

Overview

1. Regularization of DL Models
2. Optimization

A Refresher

- A model like

$$f(\mathbf{x}) = f^{(K)} \circ \cdots \circ f^{(2)} \circ f^{(1)}(\mathbf{x})$$

where $f^{(k)} : \mathbb{R}^{d_k} \to \mathbb{R}^{d_{k+1}}$ and $d_{K+1} = 1$ is called a feedforward neural network.

A Refresher

- A model like

$$f(\mathbf{x}) = f^{(K)} \circ \cdots \circ f^{(2)} \circ f^{(1)}(\mathbf{x})$$

where $f^{(k)} : \mathbb{R}^{d_k} \to \mathbb{R}^{d_{k+1}}$ and $d_{K+1} = 1$ is called a feedforward neural network.

- Each f_k refers to a ”layer”.

A Refresher

- A model like

$$f(\mathbf{x}) = f^{(K)} \circ \cdots \circ f^{(2)} \circ f^{(1)}(\mathbf{x})$$

where $f^{(k)} : \mathbb{R}^{d_k} \to \mathbb{R}^{d_{k+1}}$ and $d_{K+1} = 1$ is called a feedforward neural network.

- Each f_k refers to a "layer".

- The number of layers is called the depth.

Building blocks

- "Most" layers can be decomposed into a function like

$$f_k(\mathbf{h}) = g \circ A(\mathbf{h})$$

Building blocks

- ”Most” layers can be decomposed into a function like

$$f_k(\mathbf{h}) = g \circ A(\mathbf{h})$$

- Where g is some non-linear function (ReLU, tanh, Softmax, sigmoid/logit).

Building blocks

- ”Most” layers can be decomposed into a function like

$$f_k(\mathbf{h}) = g \circ A(\mathbf{h})$$

- Where g is some non-linear function (ReLU, tanh, Softmax, sigmoid/logit).
- And A is an *affine* transformation of the form

$$A(\mathbf{h}) = W^t\mathbf{h} + \mathbf{b}$$

Norm Penalties

- Like linear models, DL models can use ℓ_1- or ℓ_2-norm regularization.

Norm Penalties

- Like linear models, DL models can use ℓ_1- or ℓ_2-norm regularization.
- Unlike linear models, we can apply different penalties to different layers or just add one lump ℓ_1/ℓ_2 term at the end.

Norm Penalties

- Like linear models, DL models can use ℓ_1- or ℓ_2-norm regularization.
- Unlike linear models, we can apply different penalties to different layers or just add one lump ℓ_1/ℓ_2 term at the end.
- Like in the linear case, ℓ_1 tends to sparse solutions while ℓ_2 tends to minimize out-of-sample error.

Norm Penalties

- Like linear models, DL models can use ℓ_1- or ℓ_2-norm regularization.
- Unlike linear models, we can apply different penalties to different layers or just add one lump ℓ_1/ℓ_2 term at the end.
- Like in the linear case, ℓ_1 tends to sparse solutions while ℓ_2 tends to minimize out-of-sample error.
- Typically only regularize W and not $\mathbf{b}$ in the affine transformation.

Dataset Augmentation

Disclaimer: This is a technique your instructor has seen misapplied more than once.

- Limited number of samples? No problem; generate fake samples that ”look” like your original ones.

Dataset Augmentation

Disclaimer: This is a technique your instructor has seen misapplied more than once.

- Limited number of samples? No problem; generate fake samples that ”look” like your original ones.
- Classic example: shift, stretch, or rotate images in image classification.

Dataset Augmentation

Disclaimer: This is a technique your instructor has seen misapplied more than once.

- Limited number of samples? No problem; generate fake samples that ”look” like your original ones.
- Classic example: shift, stretch, or rotate images in image classification.
- Must take care that your ”faking” mechanism does not change the label or output.

Noise Injection

- Idea: Add small scale, normal noise to the weight matrices at some or all of the layers.

Noise Injection

- Idea: Add small scale, normal noise to the weight matrices at some or all of the layers.
- Effect: The winning model tends to ”find” regions of parameter space where the parameters are relatively robust against small perturbations (e.g. small changes in parameters do not lead to large changes in the loss fuction.)

Noise Injection

- Idea: Add small scale, normal noise to the weight matrices at some or all of the layers.
- Effect: The winning model tends to ”find” regions of parameter space where the parameters are relatively robust against small perturbations (e.g. small changes in parameters do not lead to large changes in the loss fuction.)
- This is a feature of DL models; for linear models, the noise will not have an effect on the optimal parameters.

Noise Injection

- Idea: Add small scale, normal noise to the weight matrices at some or all of the layers.
- Effect: The winning model tends to ”find” regions of parameter space where the parameters are relatively robust against small perturbations (e.g. small changes in parameters do not lead to large changes in the loss fuction.)
- This is a feature of DL models; for linear models, the noise will not have an effect on the optimal parameters.
- Related: Add noise to the ”label” part of the training data.

Semi-supervised Models

- Suppose you have some labeled samples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x_n}, y_n)\}$ and some unlabeled samples $\{\mathbf{x}_{n+1}, \ldots, \mathbf{x}_N\}$.

Semi-supervised Models

- Suppose you have some labeled samples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x_n}, y_n)\}$ and some unlabeled samples $\{\mathbf{x}_{n+1}, \ldots, \mathbf{x}_N\}$.
- Idea: build generative models of $P(\mathbf{x})$ and $P(\mathbf{x}, y)$ and a discriminitive model $P(y|\mathbf{x})$.

Semi-supervised Models

- Suppose you have some labeled samples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x_n}, y_n)\}$ and some unlabeled samples $\{\mathbf{x}_{n+1}, \ldots, \mathbf{x}_N\}$.
- Idea: build generative models of $P(\mathbf{x})$ and $P(\mathbf{x}, y)$ and a discriminitive model $P(y|\mathbf{x})$.
- Additional feature information coming from the unlabeled $\mathbf{x}$ samples give an advantage in classifying the y values.

Semi-supervised Models

- Suppose you have some labeled samples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x_n}, y_n)\}$ and some unlabeled samples $\{\mathbf{x}_{n+1}, \ldots, \mathbf{x}_N\}$.
- Idea: build generative models of $P(\mathbf{x})$ and $P(\mathbf{x}, y)$ and a discriminitive model $P(y|\mathbf{x})$.
- Additional feature information coming from the unlabeled $\mathbf{x}$ samples give an advantage in classifying the y values.
- Application: Named-entity recognition.

Early Stopping

- The loss function applied to a validation set tends to decrease after some iterations and then increase.

Early Stopping

- The loss function applied to a validation set tends to decrease after some iterations and then increase.
- Solution: Cache the parameters with the minimal validation loss.

Early Stopping

- The loss function applied to a validation set tends to decrease after some iterations and then increase.
- Solution: Cache the parameters with the minimal validation loss.
- Pretty much ubiquitous in DL now.

Sparsity

- We all know that ℓ_1 regularization leads to sparse parameters.

Sparsity

- We all know that ℓ_1 regularization leads to sparse parameters.
- Other approaches ensure that we can find sparse *represenations* of the original inputs.

Sparsity

- We all know that ℓ_1 regularization leads to sparse parameters.
- Other approaches ensure that we can find sparse *represenations* of the original inputs.
- Solve minimization problem:

$$\arg \min_{\mathbf{h}:\|\mathbf{h}\|_0 \leq k} \|\mathbf{x} - W\mathbf{h}\|_2^2$$

where $\|\mathbf{h}\|_0$ denotes the number of non-zero components

Matching basis pursuit

- Start with $\mathbf{x}$ and a random matrix with unit columns W.

Matching basis pursuit

- Start with $\mathbf{x}$ and a random matrix with unit columns W.
- Let $\mathbf{v}_1 = \mathbf{x}$ and then find the column $\mathbf{w}_{\mathbf{k_n}}$ of W that maximizes

$$|\langle \mathbf{v}_n, \mathbf{w}_{k_n} \rangle|$$

Matching basis pursuit

- Start with $\mathbf{x}$ and a random matrix with unit columns W.
- Let $\mathbf{v}_1 = \mathbf{x}$ and then find the column $\mathbf{w_{k_n}}$ of W that maximizes
$$|\langle \mathbf{v}_n, \mathbf{w}_{k_n} \rangle|$$
- Let $a_n = \langle \mathbf{v}_n, \mathbf{w}_{k_n}$ and
$$\mathbf{v}_{n+1} = \mathbf{v}_n - a_n \mathbf{w}_{k_n}$$

Matching basis pursuit

- Start with $\mathbf{x}$ and a random matrix with unit columns W.
- Let $\mathbf{v}_1 = \mathbf{x}$ and then find the column $\mathbf{w_{k_n}}$ of W that maximizes

$$|\langle \mathbf{v}_n, \mathbf{w}_{k_n} \rangle|$$

- Let $a_n = \langle \mathbf{v}_n, \mathbf{w}_{k_n}$ and

$$\mathbf{v}_{n+1} = \mathbf{v}_n - a_n \mathbf{w}_{k_n}$$

- Stop if $\|v_n\|_2$ is less than some threshold.

Bagging

- Idea: Similar to bootstrapping, sample k different training datasets by sampling from the original training data (with replacement).

Bagging

- Idea: Similar to bootstrapping, sample k different training datasets by sampling from the original training data (with replacement).

- Train model j on training set j.

Bagging

- Idea: Similar to bootstrapping, sample k different training datasets by sampling from the original training data (with replacement).
- Train model j on training set j.
- Ensemble the k resulting models by ”voting” on the best outcome.

Bagging

- Idea: Similar to bootstrapping, sample k different training datasets by sampling from the original training data (with replacement).
- Train model j on training set j.
- Ensemble the k resulting models by ”voting” on the best outcome.
- Downside: Computationally expensive for large networks.

Dropout

- Sample a minibatch from the training data and select a *mask* of the input features + hidden units.

Dropout

- Sample a minibatch from the training data and select a *mask* of the input features + hidden units.

- "Typical" selection is an input feature is included with probability 0.8 and hidden unit with probability 0.5.

Dropout

- Sample a minibatch from the training data and select a *mask* of the input features + hidden units.
- ”Typical” selection is an input feature is included with probability 0.8 and hidden unit with probability 0.5.
- Parameters are saved and shared among each iteration.

Dropout

- Sample a minibatch from the training data and select a *mask* of the input features + hidden units.
- ”Typical” selection is an input feature is included with probability 0.8 and hidden unit with probability 0.5.
- Parameters are saved and shared among each iteration.
- Even a relatively few draws of masks and few trainings can yield very good models.

Stochastic Gradient Descent

1. Select a sequence of learning rates (ε_k) and initial parameter set θ.

Stochastic Gradient Descent

1. Select a sequence of learning rates (ε_k) and initial parameter set θ.
2. Sample a small batch of the training data.

Stochastic Gradient Descent

1. Select a sequence of learning rates (ε_k) and initial parameter set θ.
2. Sample a small batch of the training data.
3. Compute

$$\mathbf{g} = \frac{1}{m}\nabla L(f(\mathbf{x}, \theta), y)$$

Stochastic Gradient Descent

1. Select a sequence of learning rates (ε_k) and initial parameter set θ.
2. Sample a small batch of the training data.
3. Compute
$$\mathbf{g} = \frac{1}{m}\nabla L(f(\mathbf{x}, \theta), y)$$
4. kth iteration,
$$\theta_{k+1} = \theta_k - \varepsilon_k \mathbf{g}$$

Stochastic Gradient Descent

1. Select a sequence of learning rates (ε_k) and initial parameter set θ.
2. Sample a small batch of the training data.
3. Compute
$$\mathbf{g} = \frac{1}{m}\nabla L(f(\mathbf{x}, \theta), y)$$
4. kth iteration,
$$\theta_{k+1} = \theta_k - \varepsilon_k \mathbf{g}$$

Notes: Converges if $\sum_{j=1}^{\infty} \varepsilon_j < \infty$, but tends to converge slowly (in high dimensions).

SGD with Momentum

Update SDG with the following changes:

- Choose an initial ”velocity” vector $\mathbf{v}$.

SGD with Momentum

Update SDG with the following changes:

- Choose an initial ”velocity” vector **v**.

- Choose learning rate ε and momentum parameter α.

SGD with Momentum

Update SDG with the following changes:

- Choose an initial ”velocity” vector **v**.
- Choose learning rate ε and momentum parameter α.
- Update **v** by

$$\mathbf{v}_{k+1} = \alpha \mathbf{v}_k - \varepsilon \mathbf{g}$$

SGD with Momentum

Update SDG with the following changes:

- Choose an initial "velocity" vector **v**.
- Choose learning rate ε and momentum parameter α.
- Update **v** by

$$\mathbf{v}_{k+1} = \alpha \mathbf{v}_k - \varepsilon \mathbf{g}$$

- Update θ by

$$\theta_{k+1} = \theta_k + \mathbf{v}_{k+1}$$

Adaptive Learning Rates

- AdaGrad
- RMSProp
- Adam

Goal: Keep the gradients out of local minima and exploring the parameter space efficiently.

Conjugate Gradient Descent