FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Regularization

Leonid Mill, Tobias Würfl, Katharina Breininger, Nishant Ravikumar, Sebastian Gündel, Mathis Hoffmann, Florian Thamm, Felix Denzinger
Pattern Recognition Lab, Friedrich-Alexander University of Erlangen-Nürnberg
December 3, 2018

Loss function augmentation

General outline

- Implement it **independent** of loss function
- **Only need** current weights

➔ Add it to the optimizer

- Change Neural Network container class to **gather regularization loss**
- Instead of λ we use α as a name, because lambda is a python keyword

L_2 regularization

- Forward pass:

$$\tilde{L}(\mathbf{w}) = L(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2$$

- Backward pass:

$$\mathbf{w}^{(k+1)} = \underbrace{\left(1 - \eta\lambda\right)\mathbf{w}^{(k)}}_{\text{Shrinkage}} - \eta \frac{\partial L}{\partial \mathbf{w}^{(k)}}$$

L_1 regularization

- Forward pass:

$$\tilde{L}(\mathbf{w}) = L(\mathbf{w}) + \lambda \|\mathbf{w}\|_1$$

- Backward pass:

$$\mathbf{w}^{(k+1)} = \underbrace{\mathbf{w}^{(k)} - \eta \lambda \text{sign}\left(\mathbf{w}^{(k)}\right)}_{\text{Other shrinkage}} - \eta \frac{\partial L}{\partial \mathbf{w}^{(k)}}$$

Dropout

Method

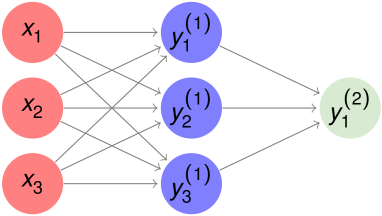

Figure: Dropout

- Implement this as a **fixed-function layer**

Method

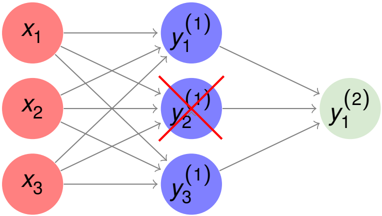

Figure: Dropout

- Implement this as a **fixed-function layer**
- Randomly set **activations** $\mapsto 0$ with probability $1 - p$

Method

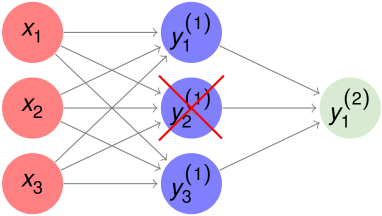

Figure: Dropout

- Implement this as a **fixed-function layer**
- Randomly set **activations** $\mapsto 0$ with probability $1 - p$
- **Test-time**: multiply activations with p

Inverted Dropout

- Can we get rid of the dropout layer at Test-time?

Inverted Dropout

- Can we get rid of the dropout layer at Test-time?

➔ change the forward-pass

- Multiply activations in forward-pass by $\frac{1}{p}$

Batch normalization

Forward pass

- Normalization as a new layer with 2 parameters, γ and β

Forward pass

➔ Normalization as a new layer with 2 parameters, γ and β

$$\tilde{\mathbf{X}} = \frac{\mathbf{X} - \boldsymbol{\mu}_B}{\sqrt{\boldsymbol{\sigma}_B^2 + \epsilon}}$$

$\boldsymbol{\mu}_B$ and $\boldsymbol{\sigma}_B$ from **mini-batch**

Forward pass

➔ Normalization as a new layer with 2 parameters, γ and $\boldsymbol{\beta}$

$$\tilde{\mathbf{X}} = \frac{\mathbf{X} - \boldsymbol{\mu}_B}{\sqrt{\boldsymbol{\sigma}_B^2 + \epsilon}}$$

$\boldsymbol{\mu}_B$ and $\boldsymbol{\sigma}_B$ from **mini-batch**

$$\hat{\mathbf{Y}} = \gamma \tilde{\mathbf{X}} + \boldsymbol{\beta}$$

Forward pass

➔ Normalization as a new layer with 2 parameters, γ and β

$$\tilde{\mathbf{X}} = \frac{\mathbf{X} - \boldsymbol{\mu}_B}{\sqrt{\boldsymbol{\sigma}_B^2 + \epsilon}}$$

$\boldsymbol{\mu}_B$ and $\boldsymbol{\sigma}_B$ from **mini-batch**

$$\hat{\mathbf{Y}} = \gamma\tilde{\mathbf{X}} + \beta$$

- $\boldsymbol{\mu}$, $\boldsymbol{\sigma}$ have the **same dimension** as the **input vectors**

Forward pass

➔ Normalization as a new layer with 2 parameters, γ and β

$$\tilde{\mathbf{X}} = \frac{\mathbf{X} - \boldsymbol{\mu}_B}{\sqrt{\boldsymbol{\sigma}_B^2 + \epsilon}}$$

$\boldsymbol{\mu}_B$ and $\boldsymbol{\sigma}_B$ from **mini-batch**

$$\hat{\mathbf{Y}} = \gamma \tilde{\mathbf{X}} + \boldsymbol{\beta}$$

- $\boldsymbol{\mu}$, $\boldsymbol{\sigma}$ have the **same dimension** as the **input vectors**
- $\boldsymbol{\beta}$, γ and $\boldsymbol{\mu}_B$, $\boldsymbol{\sigma}_B$ have same **dimension** to be able to preserve **identity**

Forward pass

➔ Normalization as a new layer with 2 parameters, γ and $\boldsymbol{\beta}$

$$\tilde{\mathbf{X}} = \frac{\mathbf{X} - \boldsymbol{\mu}_B}{\sqrt{\boldsymbol{\sigma}_B^2 + \epsilon}}$$

$\boldsymbol{\mu}_B$ and $\boldsymbol{\sigma}_B$ from **mini-batch**

$$\hat{\mathbf{Y}} = \gamma\tilde{\mathbf{X}} + \boldsymbol{\beta}$$

- $\boldsymbol{\mu}$, $\boldsymbol{\sigma}$ have the **same dimension** as the **input vectors**
- $\boldsymbol{\beta}$, γ and $\boldsymbol{\mu}_B$, $\boldsymbol{\sigma}_B$ have same **dimension** to be able to preserve **identity**
- Notice that $\boldsymbol{\beta}$ is a **bias**

Test time

- Test-time: replace $\boldsymbol{\mu}_B$ and $\boldsymbol{\sigma}_B$ with $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ of the **training set**

Test time

- Test-time: replace $\boldsymbol{\mu}_B$ and $\boldsymbol{\sigma}_B$ with $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ of the **training set**
- It's **expensive** to calculate the true training set mean and variance

Test time

- Test-time: replace $\boldsymbol{\mu}_B$ and $\boldsymbol{\sigma}_B$ with $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ of the **training set**
- It's **expensive** to calculate the true training set mean and variance
- Therefore a **moving average** is common:

$$\tilde{\boldsymbol{\mu}}^{(k)} \approx (1-\alpha)\boldsymbol{\mu}_B^{(k-1)} + \alpha\boldsymbol{\mu}_B^{(k)}$$
$$\tilde{\boldsymbol{\sigma}}^{(k)} \approx (1-\alpha)\boldsymbol{\sigma}_B^{(k-1)} + \alpha\boldsymbol{\sigma}_B^{(k)}$$

Test time

- Test-time: replace $\boldsymbol{\mu}_B$ and $\boldsymbol{\sigma}_B$ with $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ of the **training set**
- It's **expensive** to calculate the true training set mean and variance
- Therefore a **moving average** is common:

$$\tilde{\boldsymbol{\mu}}^{(k)} \approx (1-\alpha)\boldsymbol{\mu}_B^{(k-1)} + \alpha\boldsymbol{\mu}_B^{(k)}$$
$$\tilde{\boldsymbol{\sigma}}^{(k)} \approx (1-\alpha)\boldsymbol{\sigma}_B^{(k-1)} + \alpha\boldsymbol{\sigma}_B^{(k)}$$

- Moving average **decay** α (i.e. 0.8)

Backward pass

- Gradient **with respect to weights** is simply:

$$\frac{\partial L}{\partial \gamma} = \sum_{b=1}^{B} \frac{\partial L}{\partial \hat{\mathbf{Y}}_b} \tilde{\mathbf{X}}_b = \sum_{b=1}^{B} \mathbf{E}_b \tilde{\mathbf{X}}_b$$

- For the **bias** likewise we have:

$$\frac{\partial L}{\partial \beta} = \sum_{b=1}^{B} \frac{\partial L}{\partial \hat{\mathbf{Y}}_b} = \sum_{b=1}^{B} \mathbf{E}_b$$

Backward pass

The **gradient with respect to the input** is more complicated, but here it is:

$$\frac{\partial L}{\partial \tilde{\mathbf{X}}} = \frac{\partial L}{\partial \hat{\mathbf{Y}}}\gamma$$

$$\frac{\partial L}{\partial \boldsymbol{\sigma}_B^2} = \sum_{b=1}^{B} \frac{\partial L}{\partial \tilde{\mathbf{X}}_b} \cdot (\mathbf{X}_b - \boldsymbol{\mu}_B) \cdot \frac{-1}{2}\left(\boldsymbol{\sigma}_B^2 + \epsilon\right)^{\frac{-3}{2}}$$

$$\frac{\partial L}{\partial \boldsymbol{\mu}_B} = \left(\sum_{b=1}^{B} \frac{\partial L}{\partial \tilde{\mathbf{X}}_b} \cdot \frac{-1}{\sqrt{\boldsymbol{\sigma}_B^2 + \epsilon}}\right) + \underbrace{\frac{\partial L}{\partial \boldsymbol{\sigma}_B^2} \cdot \frac{\sum_{b=1}^{B} -2(\mathbf{X}_b - \boldsymbol{\mu}_B)}{B}}_{0}$$

$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \tilde{\mathbf{X}}} \cdot \frac{1}{\sqrt{\boldsymbol{\sigma}_B^2 + \epsilon}} + \frac{\partial L}{\partial \boldsymbol{\sigma}_B^2} \cdot \frac{2(\mathbf{X} - \boldsymbol{\mu}_B)}{B} + \frac{\partial L}{\partial \boldsymbol{\mu}_B} \cdot \frac{1}{B}$$

Convolutional Batch Normalization

- In CNNs batch normalization is adjusted to work **similar to convolution**

Convolutional Batch Normalization

- In CNNs batch normalization is adjusted to work **similar to convolution**
- A scalar μ , σ is calculated for the *H* **channels**

Convolutional Batch Normalization

- In CNNs batch normalization is adjusted to work **similar to convolution**
- A scalar μ , σ is calculated for the *H* **channels**
- Implementation can be reused, by observing
  - ➔ that **spatial dimensions** *M*, *N* can be treated **like the batch dimension** *B*

Convolutional Batch Normalization

- In CNNs batch normalization is adjusted to work **similar to convolution**
- A scalar μ , σ is calculated for the H **channels**
- Implementation can be reused, by observing
  - ➔ that **spatial dimensions** M, N can be treated **like the batch dimension** B
  - ➔ we can **reshape** the $B \times H \times M \times N$ tensor to a $B \times H \times M \cdot N$
  - ➔ because of our format we have to **transpose** from $B \times H \times M \cdot N$ to $B \times M \cdot N \times H$
  - ➔ and afterwards **reshape again** to have a $B \cdot M \cdot N \times H$ tensor

Convolutional Batch Normalization

- In CNNs batch normalization is adjusted to work **similar to convolution**
- A scalar μ , σ is calculated for the H **channels**
- Implementation can be reused, by observing
  - → that **spatial dimensions** M, N can be treated **like the batch dimension** B
  - → we can **reshape** the $B \times H \times M \times N$ tensor to a $B \times H \times M \cdot N$
  - → because of our format we have to **transpose** from $B \times H \times M \cdot N$ to $B \times M \cdot N \times H$
  - → and afterwards **reshape again** to have a $B \cdot M \cdot N \times H$ tensor
- Consequently we have to **reverse this** before returning the **output**

Convolutional Batch Normalization

- In CNNs batch normalization is adjusted to work **similar to convolution**
- A scalar μ , σ is calculated for the H **channels**
- Implementation can be reused, by observing
  - ➔ that **spatial dimensions** M, N can be treated **like the batch dimension** B
  - ➔ we can **reshape** the $B \times H \times M \times N$ tensor to a $B \times H \times M \cdot N$
  - ➔ because of our format we have to **transpose** from $B \times H \times M \cdot N$ to $B \times M \cdot N \times H$
  - ➔ and afterwards **reshape again** to have a $B \cdot M \cdot N \times H$ tensor
- Consequently we have to **reverse this** before returning the **output**
- ... and do the **same** in the **backward pass**

LeNet

LeNet architecture

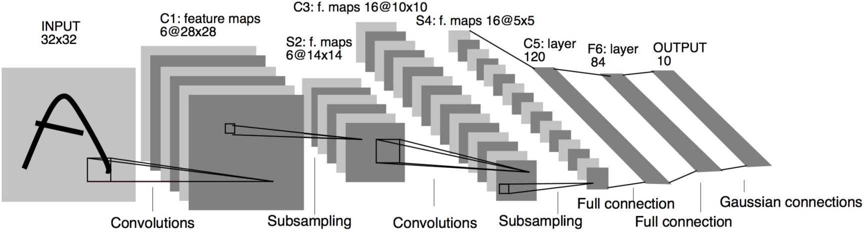

Figure: LeNet

Modified LeNet architecture

Deviations

- Input is 28×28
- Our conv only supports “same” padding - so C3 has **larger activation maps**
- Input to **C5** is also **larger**
- We only implemented ReLUs, so **no** TanH
- We also use the implemented SoftMax **instead of** RBF units

Thanks for listening.

Any questions?