Regularization

A. Maier, V. Christlein, K. Breininger, Z. Yang, L. Rist, M. Nau, S. Jaganathan, C. Liu, N. Maul, L. Folle, K. Packhäuser, M. Zinnen
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
April 24, 2023

Outline

Introduction to Regularization

Classical Techniques

Regularization in the Loss Function

Normalization

Dropout

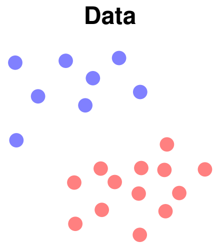
Initialization

Transfer Learning

Multi-Task Learning (MTL)

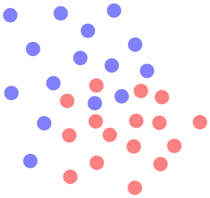
Introduction to Regularization

Fitting appropriately

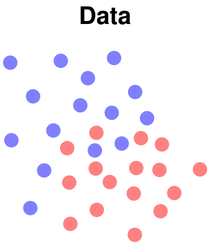


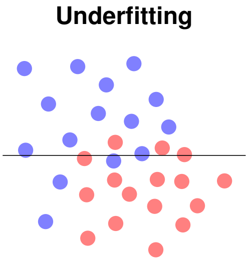
Fitting appropriately

Data



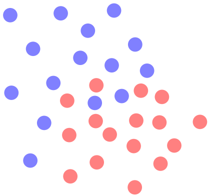
Fitting appropriately



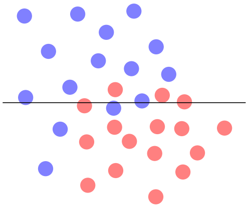


Fitting appropriately

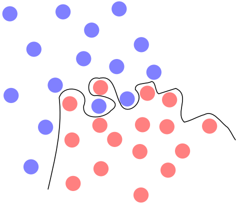
Data



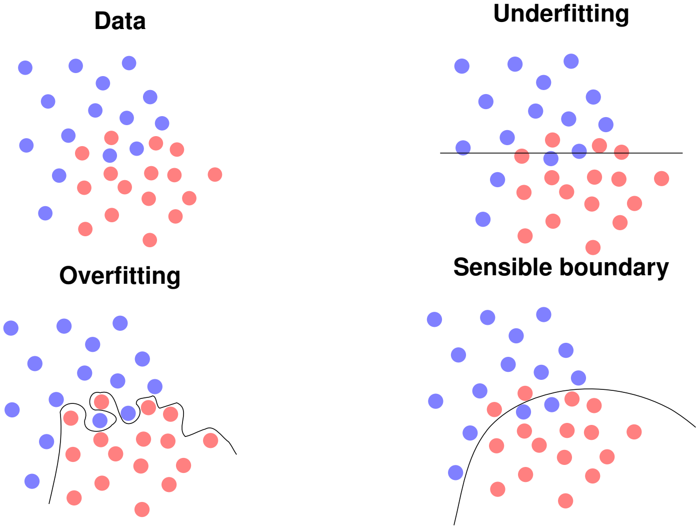
Underfitting



Overfitting



Fitting appropriately



Bias Variance Decomposition

- Regression **problem** [3] with **h**, **ideal** value from the true distribution:

$$\mathbf{h} = h(\mathbf{x}) + \epsilon\,, \quad \epsilon = \mathcal{N}(0, \sigma_\epsilon)\,.$$

Bias Variance Decomposition

- Regression **problem** [3] with **h**, **ideal** value from the true distribution:

$$\mathbf{h} = h(\mathbf{x}) + \epsilon\,, \quad \epsilon = \mathcal{N}(0, \sigma_\epsilon)\,.$$

- Using a model: $\hat{\mathbf{y}}_D = \hat{f}(\mathbf{x}|D)$ estimated from a dataset D, the expected **loss** for a single point **x** is:

$$\mathbb{E}_D\left(\ell(\mathbf{x})\right) = \mathbb{E}_D\left[(\mathbf{h} - \hat{\mathbf{y}}_D)^2\right]\,.$$

Bias Variance Decomposition

- Regression **problem** [3] with **h**, **ideal** value from the true distribution:

$$\mathbf{h} = h(\mathbf{x}) + \epsilon\,, \quad \epsilon = \mathcal{N}(0, \sigma_\epsilon)\,.$$

- Using a model: $\hat{\mathbf{y}}_D = \hat{f}(\mathbf{x}|D)$ estimated from a dataset D, the expected **loss** for a single point $\mathbf{x}$ is:

$$\mathbb{E}_D\left(\ell(\mathbf{x})\right) = \mathbb{E}_D\left[(\mathbf{h} - \hat{\mathbf{y}}_D)^2\right]\,.$$

- This can be decomposed to:

$$\mathbb{E}_D\left(\ell(\mathbf{x})\right) = \underbrace{\left(\mathbb{E}_D\left[\hat{\mathbf{y}}_D\right] - h(\mathbf{x})\right)^2}_{\text{Bias}^2} + \underbrace{\mathbb{E}_D\left[\left(\hat{\mathbf{y}}_D - \mathbb{E}_D\left[\hat{\mathbf{y}}_D\right]\right)^2\right]}_{\text{Variance}} + \underbrace{\sigma_\epsilon^2}_{\text{Irreducible Error}}\,.$$

Bias Variance Decomposition

- Regression **problem** [3] with **h**, **ideal** value from the true distribution:

$$\mathbf{h} = h(\mathbf{x}) + \epsilon\,, \quad \epsilon = \mathcal{N}(0, \sigma_\epsilon)\,.$$

- Using a model: $\hat{\mathbf{y}}_D = \hat{f}(\mathbf{x}|D)$ estimated from a dataset D, the expected **loss** for a single point $\mathbf{x}$ is:

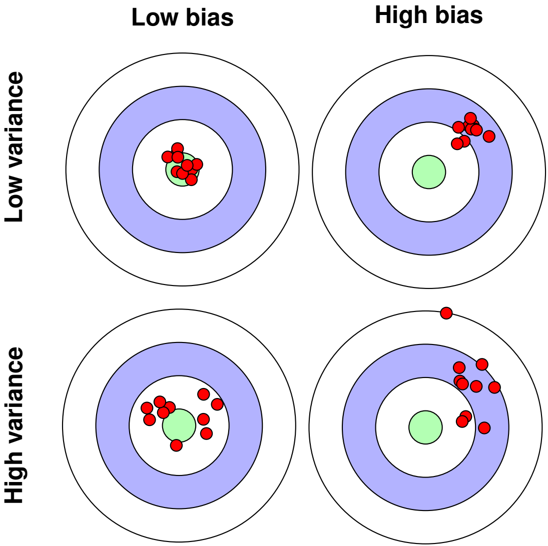
$$\mathbb{E}_D\left(\ell(\mathbf{x})\right) = \mathbb{E}_D\left[(\mathbf{h} - \hat{\mathbf{y}}_D)^2\right]\,.$$

- This can be decomposed to:

$$\mathbb{E}_D\left(\ell(\mathbf{x})\right) = \underbrace{\left(\mathbb{E}_D\left[\hat{\mathbf{y}}_D\right] - h(\mathbf{x})\right)^2}_{\text{Bias}^2} + \underbrace{\mathbb{E}_D\left[\left(\hat{\mathbf{y}}_D - \mathbb{E}_D\left[\hat{\mathbf{y}}_D\right]\right)^2\right]}_{\text{Variance}} + \underbrace{\sigma_\epsilon^2}_{\text{Irreducible Error}}\,.$$

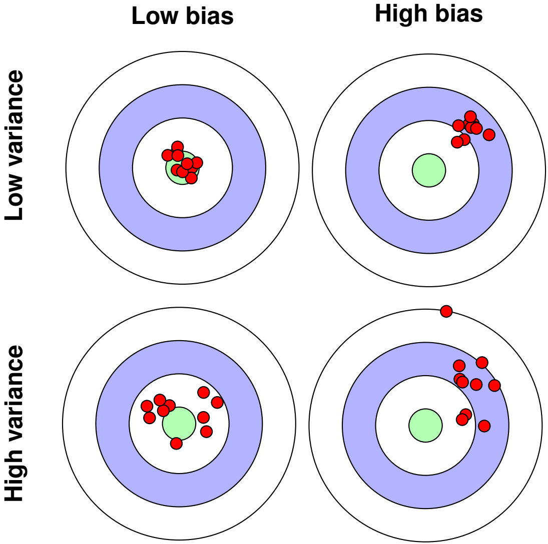
- Integrating this over every datapoint $\mathbf{x}$ we get $L_D(\mathbf{X})$ from the $\ell_D(\mathbf{x})$.
- A similar decomposition exists for classification using the zero-one loss [9, pp. 468-471].

Bias Variance Tradeoff



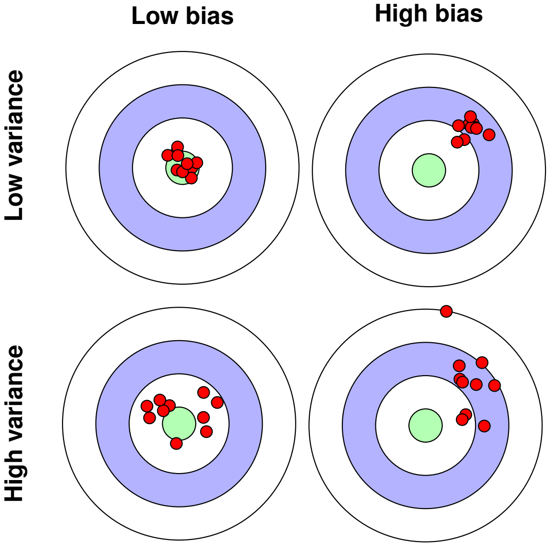
- We'd like to **minimize bias and variance**

Bias Variance Tradeoff



- We'd like to **minimize bias and variance**
- However, if we choose a type of model for a given dataset:
  - → Simultaneously optimizing **bias** and **variance** is **impossible** in general

Bias Variance Tradeoff



- We'd like to **minimize bias and variance**
- However, if we choose a type of model for a given dataset:
  - → Simultaneously optimizing **bias** and **variance** is **impossible** in general
- Bias and variance can be studied together as **model capacity**

Model Capacity

- **Capacity** of a model $\mapsto$ **variety of functions** it can approximate
- **Related** to number of parameters but **by far not equal**

Model Capacity

- **Capacity** of a model $\mapsto$ **variety of functions** it can approximate
- **Related** to number of parameters but **by far not equal**
- Vapnik-Chervonenkis (VC) dimension provides a **measure** of capacity
  - ➔ Based on **counting** how many **points** can be **separated** by a model

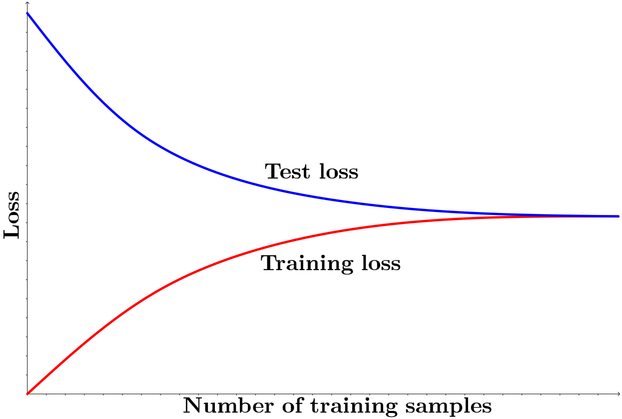
Model Capacity

- **Capacity** of a model $\mapsto$ **variety of functions** it can approximate
- **Related** to number of parameters but **by far not equal**
- Vapnik-Chervonenkis (VC) dimension provides a **measure** of capacity
  - ➔ Based on **counting** how many **points** can be **separated** by a model
  - VC dimension of neural networks is usually **extremely high** compared to classical methods
  - Remember the paper learning ImageNet with random labels? [18]

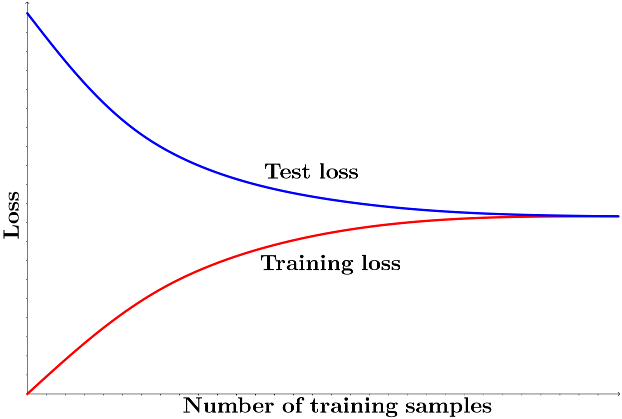
Model Capacity

- **Capacity** of a model $\mapsto$ **variety of functions** it can approximate
- **Related** to number of parameters but **by far not equal**
- Vapnik-Chervonenkis (VC) dimension provides a **measure** of capacity
  - ➔ Based on **counting** how many **points** can be **separated** by a model
  - VC dimension of neural networks is usually **extremely high** compared to classical methods
  - Remember the paper learning ImageNet with random labels? [18]
  - ➔ VC dimension is **ineffective** in judging the **real capacity** of neural networks
- We can always reduce the **bias** by $\uparrow$ the **model capacity**

The Role of Data

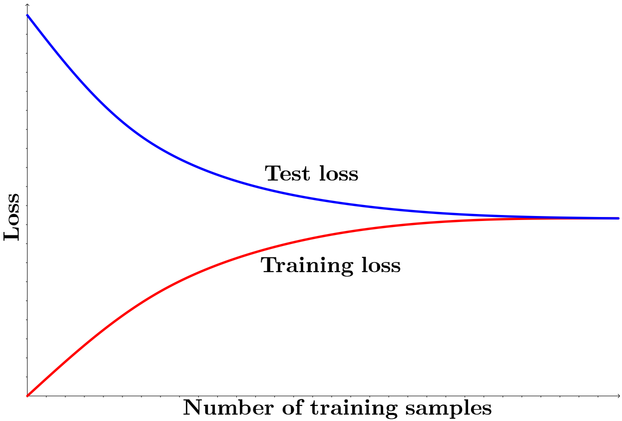


The Role of Data



- The **variance** can be optimized by using **more training data**

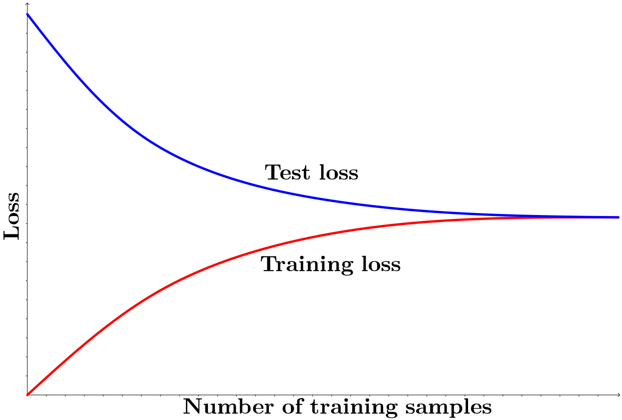
The Role of Data



- The **variance** can be optimized by using **more training data**

➔ The model capacity has to **match** the **size** of the **training set**

The Role of Data

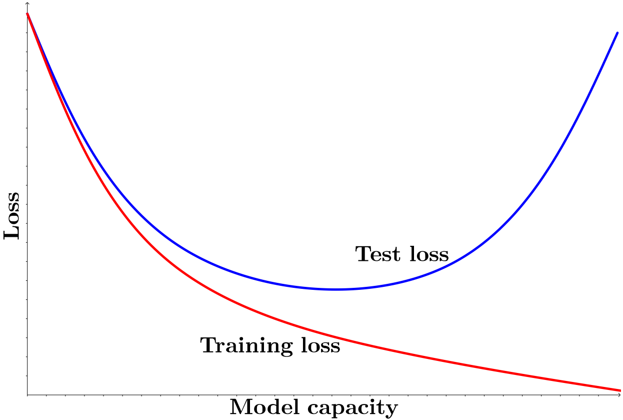


- The **variance** can be optimized by using **more training data**

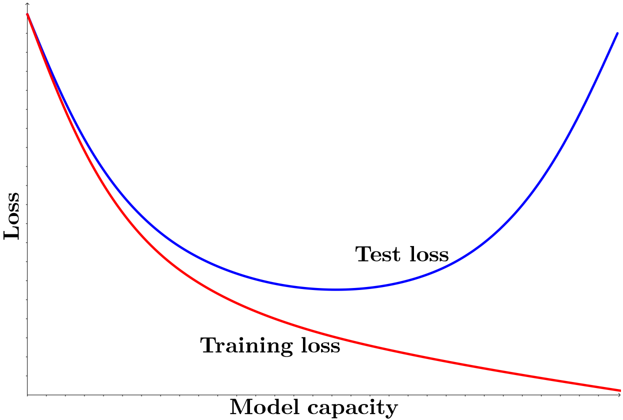
➔ The model capacity has to **match** the **size** of the **training set**

- What if we can't acquire more data?

Finite Dataset

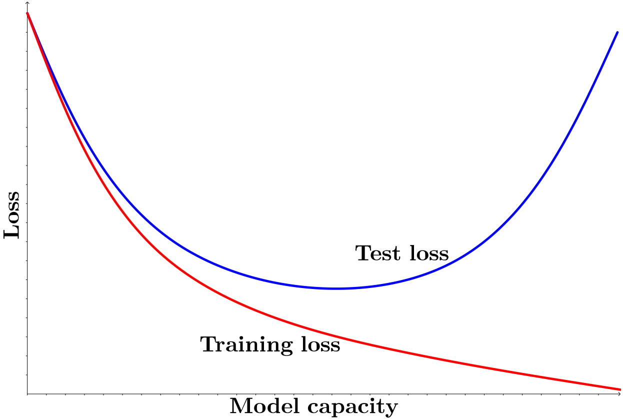


Finite Dataset



- We can trade an ↑ in **bias** for ↓ in **variance**

Finite Dataset



- We can trade an $\uparrow$ in **bias** for $\downarrow$ in **variance**

➔ For a **specific problem** there might be favorable tradeoffs!

Regularization reduces Overfitting

How can we find such a favorable tradeoff?

→ **By enforcing prior knowledge**

Regularization reduces Overfitting

How can we find such a favorable tradeoff?

→ **By enforcing prior knowledge**

Model prior knowledge

- Augment data
- Adapt architecture
- Adapt training process
- Preprocessing

Regularization reduces Overfitting

How can we find such a favorable tradeoff?

→ **By enforcing prior knowledge**

Model prior knowledge

- Augment data
- Adapt architecture
- Adapt training process
- Preprocessing

Actual regularizers

- Equality constraints
- Inequality constraints

NEXT TIME
ON DEEP LEARNING

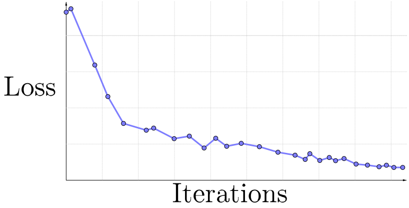
Regularization - Part 2

A. Maier, V. Christlein, K. Breininger, Z. Yang, L. Rist, M. Nau, S. Jaganathan, C. Liu, N. Maul, L. Folle, K. Packhäuser, M. Zinnen
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
April 24, 2023

Classical Techniques

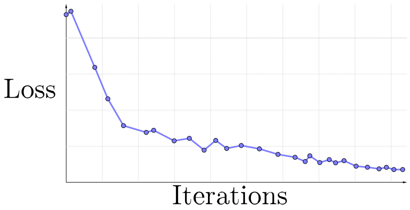
Overfitting on Learning Curve

Training set

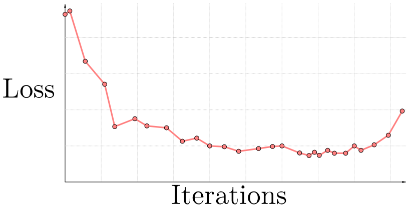


Overfitting on Learning Curve

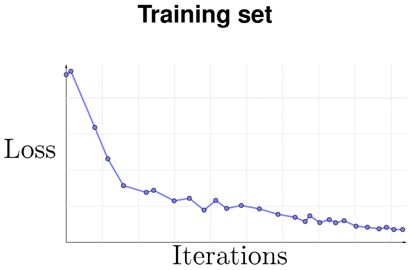
Training set

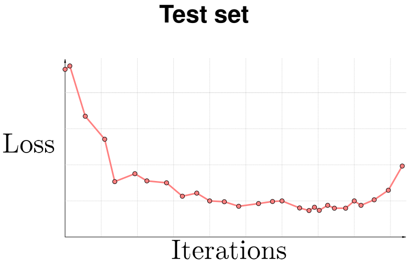


Test set



Overfitting on Learning Curve

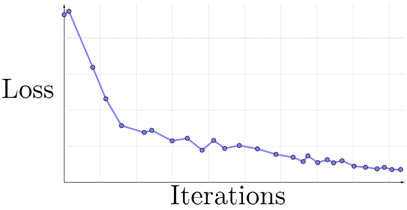




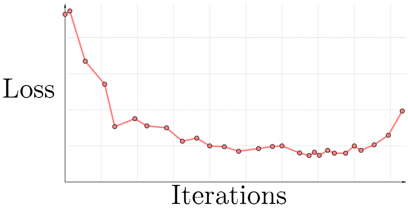
- Test set **must not** be used for training!

Overfitting on Learning Curve

Training set



Test set

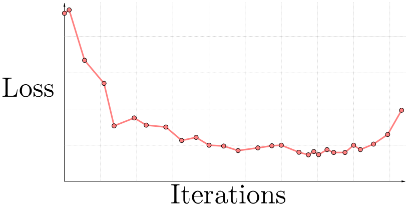


- Test set **must not** be used for training!

➔ Split of **validation set** from training data

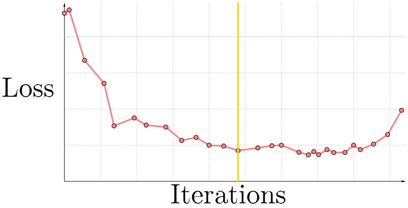
Early Stopping

Validation set



Early Stopping

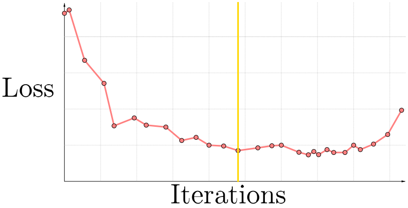
Validation set



- Define stopping criterion

Early Stopping

Validation set



- Define stopping criterion

→ Use parameters with **minimum validation loss**

Data Augmentation

- **Artificially enlarge dataset**

Data Augmentation

→ **Artificially enlarge dataset**

- But how?

Data Augmentation

→ **Artificially enlarge dataset**

- But how?
- Every **transformation** which the **label** should be **invariant** to:

Probably the same class:

Data Augmentation

→ **Artificially enlarge dataset**

- But how?
- Every **transformation** which the **label** should be **invariant** to:

Probably the same class:

Still you have to be careful!

Data Augmentation

Common transformations:

Data Augmentation

Common transformations:

1. random spatial transformations:
   - affine transformations
   - elastic transformations
   - ...

Data Augmentation

Common transformations:

1. random spatial transformations:
   - affine transformations
   - elastic transformations
   - ...
2. pixel transformations
   - changing resolution
   - random noise
   - changing pixel distribution
   - ...

Regularization in the Loss Function

Maximum A Posteriori Estimation

→ **Bayesian** approach considering the weights uncertain: $\mathbf{w} \mapsto p(\mathbf{w})$

- and we have a dataset $\mathbf{X}$ with associated labels $\mathbf{Y}$

Maximum A Posteriori Estimation

➔ **Bayesian** approach considering the weights uncertain: $\mathbf{w} \mapsto p(\mathbf{w})$

- and we have a dataset $\mathbf{X}$ with associated labels $\mathbf{Y}$
- Now we can state:

$$p(\mathbf{w}, \mathbf{Y}, \mathbf{X}) = p(\mathbf{w}|\mathbf{Y}, \mathbf{X})\, p(\mathbf{Y}, \mathbf{X})$$

Maximum A Posteriori Estimation

- ➔ **Bayesian** approach considering the weights uncertain: $\mathbf{w} \mapsto p(\mathbf{w})$
- and we have a dataset $\mathbf{X}$ with associated labels $\mathbf{Y}$
- Now we can state:

$$\begin{aligned} p(\mathbf{w}, \mathbf{Y}, \mathbf{X}) &= p(\mathbf{w}|\mathbf{Y}, \mathbf{X})\, p(\mathbf{Y}, \mathbf{X}) \\ &= p(\mathbf{Y}|\mathbf{X}, \mathbf{w})\, p(\mathbf{X}, \mathbf{w}) \end{aligned}$$

Maximum A Posteriori Estimation

➔ **Bayesian** approach considering the weights uncertain: $\mathbf{w} \mapsto p(\mathbf{w})$

- and we have a dataset **X** with associated labels **Y**
- Now we can state:

$$\begin{aligned} p(\mathbf{w}, \mathbf{Y}, \mathbf{X}) &= p(\mathbf{w}|\mathbf{Y}, \mathbf{X})\, p(\mathbf{Y}, \mathbf{X}) \\ &= p(\mathbf{Y}|\mathbf{X}, \mathbf{w})\, p(\mathbf{X}, \mathbf{w}) \\ p(\mathbf{w}|\mathbf{Y}, \mathbf{X}) &= \frac{p(\mathbf{Y}|\mathbf{X}, \mathbf{w})\, p(\mathbf{X}, \mathbf{w})}{p(\mathbf{Y}, \mathbf{X})} \end{aligned}$$

Maximum A Posteriori Estimation

➔ **Bayesian** approach considering the weights uncertain: $\mathbf{w} \mapsto p(\mathbf{w})$

- and we have a dataset **X** with associated labels **Y**
- Now we can state:

$$\begin{aligned} p(\mathbf{w}, \mathbf{Y}, \mathbf{X}) &= p(\mathbf{w}|\mathbf{Y}, \mathbf{X})\, p(\mathbf{Y}, \mathbf{X}) \\ &= p(\mathbf{Y}|\mathbf{X}, \mathbf{w})\, p(\mathbf{X}, \mathbf{w}) \end{aligned}$$

$$p(\mathbf{w}|\mathbf{Y}, \mathbf{X}) = \frac{p(\mathbf{Y}|\mathbf{X}, \mathbf{w})\, p(\mathbf{X}, \mathbf{w})}{p(\mathbf{Y}, \mathbf{X})}$$

$$p(\mathbf{w}|\mathbf{Y}, \mathbf{X}) = \frac{p(\mathbf{Y}|\mathbf{X}, \mathbf{w})\, p(\mathbf{X})\, p(\mathbf{w})}{p(\mathbf{Y}|\mathbf{X})p(\mathbf{X})}$$

Maximum A Posteriori Estimation

- **Bayesian** approach considering the weights uncertain: $\mathbf{w} \mapsto p(\mathbf{w})$
- and we have a dataset $\mathbf{X}$ with associated labels $\mathbf{Y}$
- Now we can state:

$$\begin{aligned} p(\mathbf{w}, \mathbf{Y}, \mathbf{X}) &= p(\mathbf{w}|\mathbf{Y}, \mathbf{X})\, p(\mathbf{Y}, \mathbf{X}) \\ &= p(\mathbf{Y}|\mathbf{X}, \mathbf{w})\, p(\mathbf{X}, \mathbf{w}) \end{aligned}$$

$$p(\mathbf{w}|\mathbf{Y}, \mathbf{X}) = \frac{p(\mathbf{Y}|\mathbf{X}, \mathbf{w})\, p(\mathbf{X}, \mathbf{w})}{p(\mathbf{Y}, \mathbf{X})}$$

$$p(\mathbf{w}|\mathbf{Y}, \mathbf{X}) = \frac{p(\mathbf{Y}|\mathbf{X}, \mathbf{w})\, p(\mathbf{X})\, p(\mathbf{w})}{p(\mathbf{Y}|\mathbf{X}) p(\mathbf{X})}$$

- Because $p(\mathbf{Y}|\mathbf{X})$ does not depend on $\mathbf{w}$, and $\mathbf{X}$ and $\mathbf{w}$ are independent, we can obtain an estimate as:

$$\text{MAP}(\mathbf{w}) := \underset{\mathbf{w}}{\text{maximize}} \quad \left\{ p(\mathbf{Y}|\mathbf{X}, \mathbf{w})\, p(\mathbf{w}) \right\}$$

Maximum A Posteriori Estimation

→ **Bayesian** approach considering the weights uncertain: $\mathbf{w} \mapsto p(\mathbf{w})$

- and we have a dataset $\mathbf{X}$ with associated labels $\mathbf{Y}$
- Now we can state:

$$\begin{aligned} p(\mathbf{w}, \mathbf{Y}, \mathbf{X}) &= p(\mathbf{w}|\mathbf{Y}, \mathbf{X})\, p(\mathbf{Y}, \mathbf{X}) \\ &= p(\mathbf{Y}|\mathbf{X}, \mathbf{w})\, p(\mathbf{X}, \mathbf{w}) \\ p(\mathbf{w}|\mathbf{Y}, \mathbf{X}) &= \frac{p(\mathbf{Y}|\mathbf{X}, \mathbf{w})\, p(\mathbf{X}, \mathbf{w})}{p(\mathbf{Y}, \mathbf{X})} \\ p(\mathbf{w}|\mathbf{Y}, \mathbf{X}) &= \frac{p(\mathbf{Y}|\mathbf{X}, \mathbf{w})\, p(\mathbf{X})\, p(\mathbf{w})}{p(\mathbf{Y}|\mathbf{X})p(\mathbf{X})} \end{aligned}$$

- Because $p(\mathbf{Y}|\mathbf{X})$ does not depend on **w**, and **X** and **w** are independent, we can obtain an estimate as:

$$\mathrm{MAP}(\mathbf{w}) := \underbrace{\underset{\mathbf{w}}{\text{maximize}} \quad \left\{ p(\mathbf{Y}|\mathbf{X}, \mathbf{w})\right.}_{\text{MLE estimator}} \underbrace{p(\mathbf{w})}_{\text{Prior}} \left.\right\}$$

Augmenting the Loss Function

Enforce

Augmenting the Loss Function

Enforce ...small norm:

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

Augmenting the Loss Function

Enforce ...small norm:

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

When λ is positive, we can identify this as the Lagrangian function of:

$$\underset{\mathbf{w}}{\text{minimize}} \quad \left\{ L(\mathbf{w}, \mathbf{X}, \mathbf{Y}) \right\} \quad s.t.: \; \|\mathbf{w}\|_2^2 \leq \alpha$$

with an unknown data-dependent α.

Augmenting the Loss Function

Enforce ...small norm:

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

When λ is positive, we can identify this as the Lagrangian function of:

$$\underset{\mathbf{w}}{\text{minimize}} \quad \left\{ L(\mathbf{w}, \mathbf{X}, \mathbf{Y}) \right\} \quad s.t.: \ \|\mathbf{w}\|_2^2 \leq \alpha$$

with an unknown data-dependent α.

Backpropagation of the augmented loss:

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

- Often called “weight decay”

Augmenting the Loss Function

Enforce ...small norm:

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

When λ is positive, we can identify this as the Lagrangian function of:

$$\underset{\mathbf{w}}{\text{minimize}} \quad \left\{ L(\mathbf{w}, \mathbf{X}, \mathbf{Y}) \right\} \quad s.t.: \; \|\mathbf{w}\|_2^2 \leq \alpha$$

with an unknown data-dependent α.

Backpropagation of the augmented loss:

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

- Often called “weight decay”
- If we optimize the **training-loss** for λ we will usually receive $\lambda = 0$.

Augmenting the Loss Function

Enforce ...small norm:

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

When λ is positive, we can identify this as the Lagrangian function of:

$$\underset{\mathbf{w}}{\text{minimize}} \quad \left\{ L(\mathbf{w}, \mathbf{X}, \mathbf{Y}) \right\} \quad s.t.: \ \|\mathbf{w}\|_2^2 \leq \alpha$$

with an unknown data-dependent α.

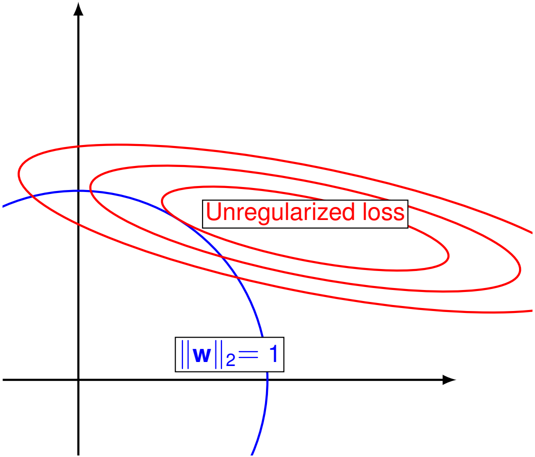
Backpropagation of the augmented loss:

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

- Often called “weight decay”
- If we optimize the **training-loss** for λ we will usually receive $\lambda = 0$.

➔ We trade increased **bias** for reduced **variance**

Graphical Effect of L_2 Regularization



Augmenting the Loss Function (cont.)

Enforce

Augmenting the Loss Function (cont.)

Enforce ...sparsity:

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

Augmenting the Loss Function (cont.)

Enforce ...sparsity:

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

When λ is positive, we can identify this as the Lagrangian function of:

$$\underset{\mathbf{w}}{\text{minimize}} \quad \left\{ L(\mathbf{w}, \mathbf{X}, \mathbf{Y}) \right\} \quad s.t.: \ \|\mathbf{w}\|_1 \leq \alpha \,,$$

with an unknown data-dependent α.

Augmenting the Loss Function (cont.)

Enforce ...sparsity:

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

When λ is positive, we can identify this as the Lagrangian function of:

$$\underset{\mathbf{w}}{\text{minimize}} \quad \left\{ L(\mathbf{w}, \mathbf{X}, \mathbf{Y}) \right\} \quad s.t.: \ \|\mathbf{w}\|_1 \leq \alpha \,,$$

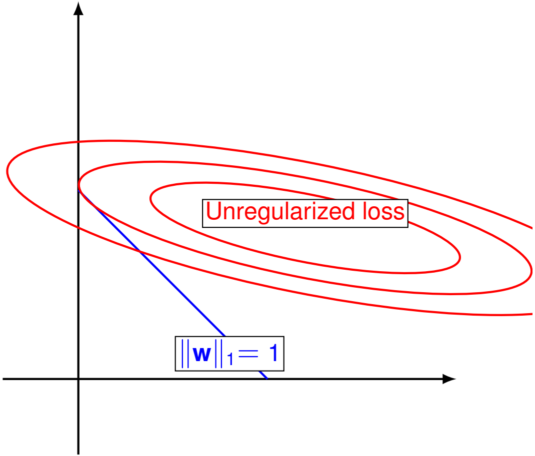
with an unknown data-dependent α.

Backpropagation of the augmented loss with the corresponding **subgradient**:

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

➔ Same as before

Graphical Effect (cont.)



Norm constraints

Can't we just set a limit on a norm of the weights?

Norm constraints

Can't we just set a limit on a norm of the weights?

Enforce ... norm below a maximum:

$$\underset{\mathbf{w}}{\text{minimize}} \quad \left\{ L(\mathbf{w}, \mathbf{X}, \mathbf{Y}) \right\} \quad s.t.: \ \|\mathbf{w}\| \leq \alpha \,, \quad \alpha > 0$$

Norm constraints

Can't we just set a limit on a norm of the weights?

Enforce ... norm below a maximum:

$$\underset{\mathbf{w}}{\text{minimize}} \quad \left\{ L(\mathbf{w}, \mathbf{X}, \mathbf{Y}) \right\} \quad s.t. : \|\mathbf{w}\| \leq \alpha\,, \quad \alpha > 0$$

→ Perform parameter update, then project to unit-"ball"

- Still a kind of shrinkage
- Prohibits **exploding gradients** but also hides it

Other Variants of Changing the Loss

- We can have a **constraint** and a λ **individual** for every layer

➔ Possible but **no gains** are reported

Other Variants of Changing the Loss

- We can have a **constraint** and a λ **individual** for every layer

→ Possible but **no gains** are reported

- Instead of the weights, **activations** can be **constrained**

Other Variants of Changing the Loss

- We can have a **constraint** and a λ **individual** for every layer

➔ Possible but **no gains** are reported

- Instead of the weights, **activations** can be **constrained**
- Different variants have been used for **sparse autoencoders**
- We will cover this when we talk about unsupervised learning

NEXT TIME

ON DEEP LEARNING

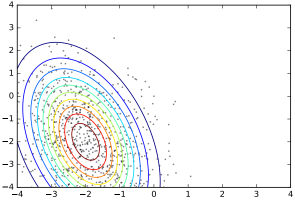
Regularization - Part 3

A. Maier, V. Christlein, K. Breininger, Z. Yang, L. Rist, M. Nau, S. Jaganathan, C. Liu, N. Maul, L. Folle, K. Packhäuser, M. Zinnen
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
April 24, 2023

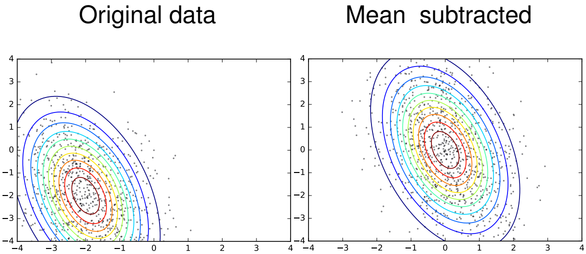
Normalization

Data Normalization

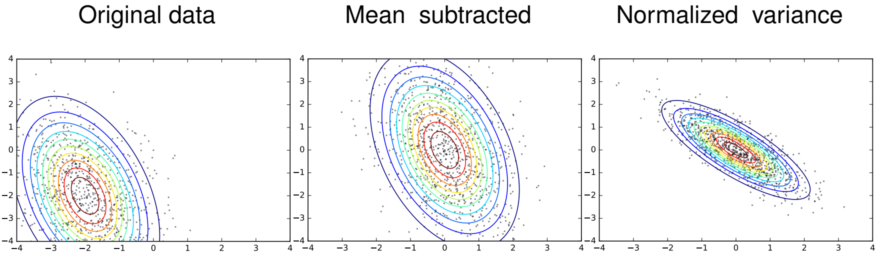
Original data



Data Normalization

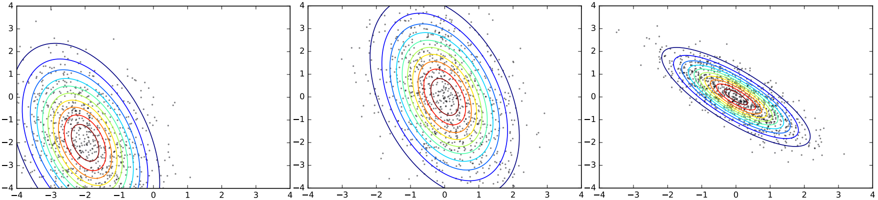


Data Normalization



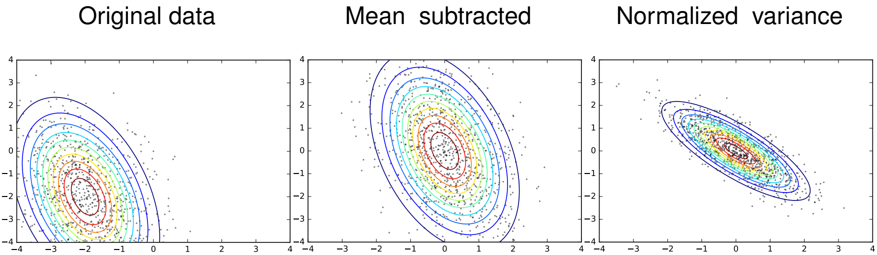
Data Normalization

Original data | Mean subtracted | Normalized variance



- Normalization: min/max or **variance**
- Only use **training data** to calculate normalization!

Data Normalization



- Normalization: min/max or **variance**
- Only use **training data** to calculate normalization!

- Normalization of input data
- Normalization **within** the network

Batch normalization (BN)

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

Batch normalization (BN)

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

$$\tilde{x}_i = \frac{x_i - \mu_{B,i}}{\sqrt{\sigma_{B,i}^2 + \epsilon}}$$

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

Batch normalization (BN)

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

$$\tilde{x}_i = \frac{x_i - \mu_{B,i}}{\sqrt{\sigma^2_{B,i} + \epsilon}}$$

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

$$\hat{y}_i = \gamma_i \tilde{x}_i + \beta_i$$

Batch normalization (BN)

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

$$\tilde{x}_i = \frac{x_i - \mu_{B,i}}{\sqrt{\sigma_{B,i}^2 + \epsilon}}$$

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

$$\hat{y}_i = \gamma_i \tilde{x}_i + \beta_i$$

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

Batch normalization (BN)

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

$$\tilde{x}_i = \frac{x_i - \mu_{B,i}}{\sqrt{\sigma^2_{B,i} + \epsilon}}$$

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

$$\hat{y}_i = \gamma_i \tilde{x}_i + \beta_i$$

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

- So the $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are **vectors, of the same dimension** as the **activation** vector.

Batch normalization (BN)

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

$$\tilde{x}_i = \frac{x_i - \mu_{B,i}}{\sqrt{\sigma^2_{B,i} + \epsilon}}$$

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

$$\hat{y}_i = \gamma_i \tilde{x}_i + \beta_i$$

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

- So the $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are **vectors, of the same dimension** as the **activation** vector.
- Paired with **convolutional** layers batch normalization is **different** by computing a **scalar** μ and σ for every **channel**.

Why does batch normalization help?

- Overwhelming **practical evidence** that batch normalization helps
- Original motivation by Ioffe and Szegedy [1]:
  BN reduces **Internal Covariate Shift**

Why does batch normalization help?

- Overwhelming **practical evidence** that batch normalization helps
- Original motivation by Ioffe and Szegedy [1]:
  BN reduces **Internal Covariate Shift**

Internal Covariate Shift

- ReLU is not **zero-centered**
- Initialization and input distribution might not be normalized

➔ Input distribution **shifts** over time

- **Deeper nets** $\mapsto$ amplified effect

Why does batch normalization help?

- Overwhelming **practical evidence** that batch normalization helps
- Original motivation by Ioffe and Szegedy [1]:
  BN reduces **Internal Covariate Shift**

Internal Covariate Shift

- ReLU is not **zero-centered**
- Initialization and input distribution might not be normalized

➔ Input distribution **shifts** over time

- **Deeper nets** $\mapsto$ amplified effect
- Layers constantly adapt

Why does batch normalization help?

- Overwhelming **practical evidence** that batch normalization helps
- Original motivation by Ioffe and Szegedy [1]:
  BN reduces **Internal Covariate Shift**

Internal Covariate Shift

- ReLU is not **zero-centered**
- Initialization and input distribution might not be normalized

➔ Input distribution **shifts** over time

- **Deeper nets** $\mapsto$ amplified effect
- Layers constantly adapt

➔ **Slow learning**

Why does batch normalization help?

- Overwhelming **practical evidence** that batch normalization helps
- Original motivation by Ioffe and Szegedy [1]:
  BN reduces **Internal Covariate Shift**

Internal Covariate Shift

- ReLU is not **zero-centered**
- Initialization and input distribution might not be normalized

➔ Input distribution **shifts** over time

- **Deeper nets** $\mapsto$ amplified effect
- Layers constantly adapt

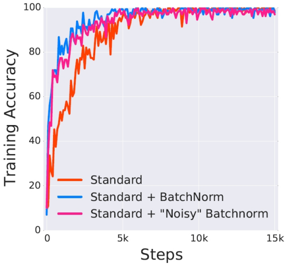
➔ **Slow learning**

- **But**: Little concrete evidence to support this theory

Why does batch normalization help? (cont.)

NeurIPS 2018: How Does Batch Normalization Help Optimization? [19]:

- BN helps **even** if internal covariate shift is introduced again afterwards

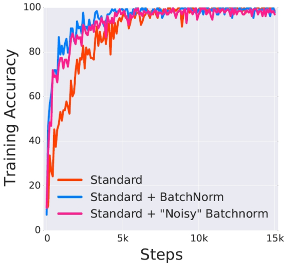


Source: [19]

Why does batch normalization help? (cont.)

NeurIPS 2018: How Does Batch Normalization Help Optimization? [19]:

- BN helps **even** if internal covariate shift is introduced again afterwards
- Experimental evidence that BN **smooths the loss landscape**

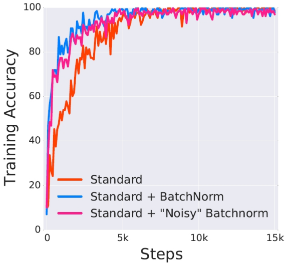


Source: [19]

Why does batch normalization help? (cont.)

NeurIPS 2018: How Does Batch Normalization Help Optimization? [19]:

- BN helps **even** if internal covariate shift is introduced again afterwards
- Experimental evidence that BN **smooths the loss landscape**
- Reminder: function f is L-Lipschitz if
  $|f(x_1) - f(x_2)| \leq L||x_1 - x_2|| \quad \forall x_1, x_2$

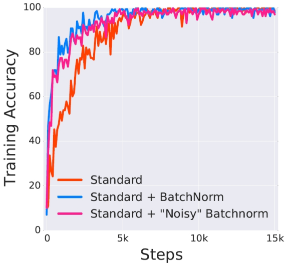


Source: [19]

Why does batch normalization help? (cont.)

NeurIPS 2018: How Does Batch Normalization Help Optimization? [19]:

- BN helps **even** if internal covariate shift is introduced again afterwards
- Experimental evidence that BN **smooths the loss landscape**
- Reminder: function f is L-Lipschitz if $|f(x_1) - f(x_2)| \leq L||x_1 - x_2|| \quad \forall x_1, x_2$
- Proof that BN improves Lipschitzness
  - of the loss function
  - of the gradient ➔ gradient is more predictive

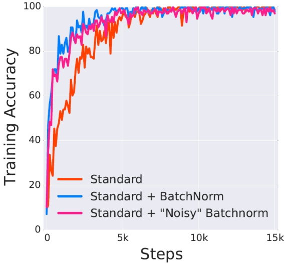


Source: [19]

Why does batch normalization help? (cont.)

NeurIPS 2018: How Does Batch Normalization Help Optimization? [19]:

- BN helps **even** if internal covariate shift is introduced again afterwards
- Experimental evidence that BN **smooths the loss landscape**
- Reminder: function f is L-Lipschitz if $|f(x_1) - f(x_2)| \leq L||x_1 - x_2|| \quad \forall x_1, x_2$
- Proof that BN improves Lipschitzness
  - of the loss function
  - of the gradient ➔ gradient is more predictive



- BN improves stability e.g. with respect to hyperparameters, initialization, convergence
- Similar properties also observed for l_p-normalization

Source: [19]

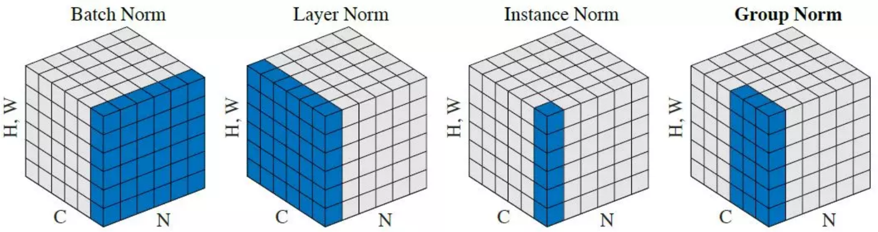
Generalizations

$$\hat{y} = f\left(g\frac{x-\mu}{\sigma} + b\right)$$

y: output
f: activation function
x: input
g,b: adaptive gain and bias
μ,σ: mean and std dev.

Calculating μ, σ over

- ... activations of a batch [1], a layer [15], spatial dims [13], a group [11]
- ... weights of a layer [20]



Source: Wu et al. [11]

Self Normalizing Neural Networks

- Method that **addresses** the **stability problem** of SGD [14]

Self Normalizing Neural Networks

- Method that **addresses** the **stability problem** of SGD [14]
- Key element: **SeLU** + specific weight initialization

$$f(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{else} \end{cases}$$

  With $\lambda_{01} = 1.0507$ and $\alpha_{01} = 1.6733$ for $\mu = 0$ and $\sigma = 1$
- Guarantee that μ and the σ of the activations stays near 0 and 1 through the network

  ➔ Stable activations, stable training

Self Normalizing Neural Networks

- Method that **addresses** the **stability problem** of SGD [14]
- Key element: **SeLU** + specific weight initialization

$$f(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{else} \end{cases}$$

With $\lambda_{01} = 1.0507$ and $\alpha_{01} = 1.6733$ for $\mu = 0$ and $\sigma = 1$

- Guarantee that μ and the σ of the activations stays near 0 and 1 through the network

  ➔ Stable activations, stable training
- Require special form of drop-out: Instead of dropping to zero, drop to

$$-\lambda \cdot \alpha$$

- Additionally they use an **affine transform** to restore variance and mean

Self Normalizing Neural Networks

- Method that **addresses** the **stability problem** of SGD [14]
- Key element: **SeLU** + specific weight initialization

$$f(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{else} \end{cases}$$

With $\lambda_{01} = 1.0507$ and $\alpha_{01} = 1.6733$ for $\mu = 0$ and $\sigma = 1$

- Guarantee that μ and the σ of the activations stays near 0 and 1 through the network

  ➔ Stable activations, stable training
- Require special form of drop-out: Instead of dropping to zero, drop to

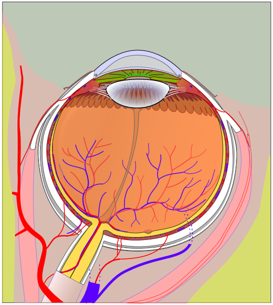
$$-\lambda \cdot \alpha$$

- Additionally they use an **affine transform** to restore variance and mean
- The **SNN** concept resembles **batch normalization**

FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Dropout

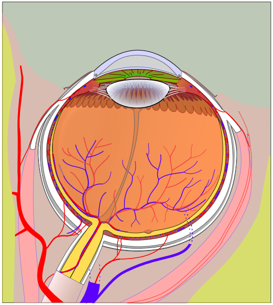
Co-adaptation

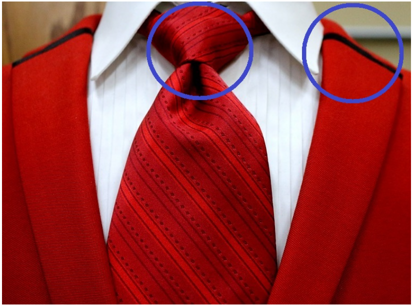


All elements **depend** on each other

Source: Jordi March / CC-BY-SA-3.0

Co-adaptation

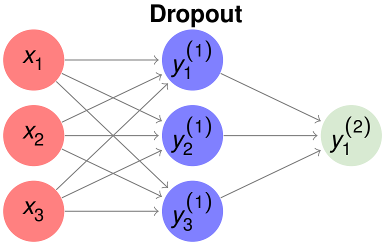




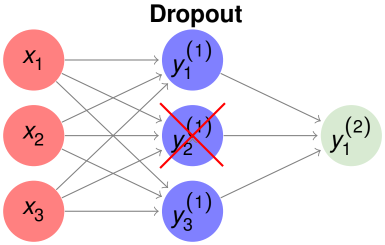
All elements **depend** on each other

➔Contrary to **independent** features

Dropout and Dropconnect



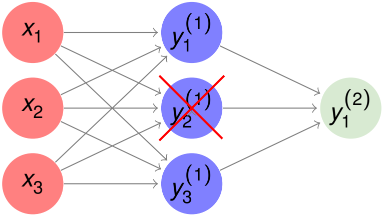
Dropout and Dropconnect



- Randomly set activations to zero with probability $1 - p$

Dropout and Dropconnect

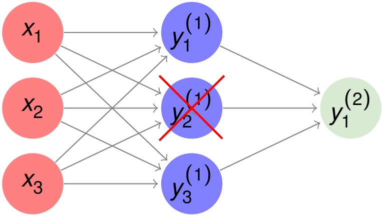
Dropout



- Randomly set activations to zero with probability $1 - p$
- Test-time: multiply activation with p

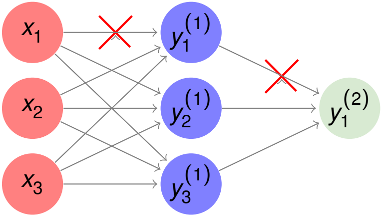
Dropout and Dropconnect

Dropout



- Randomly set activations to zero with probability $1 - p$
- Test-time: multiply activation with p

Dropconnect



- Generalizes Dropout
- Less efficient implementation (masking)

NEXT TIME

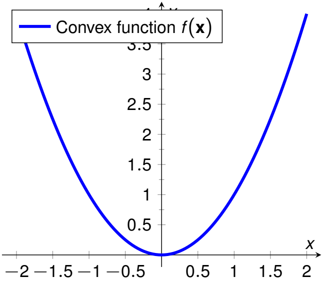
ON DEEP LEARNING

Regularization - Part 4

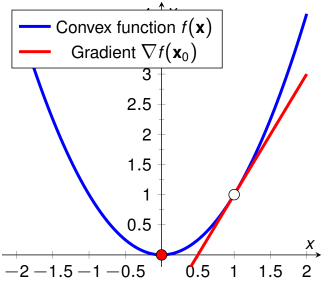
A. Maier, V. Christlein, K. Breininger, Z. Yang, L. Rist, M. Nau, S. Jaganathan, C. Liu, N. Maul, L. Folle, K. Packhäuser, M. Zinnen
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
April 24, 2023

Initialization

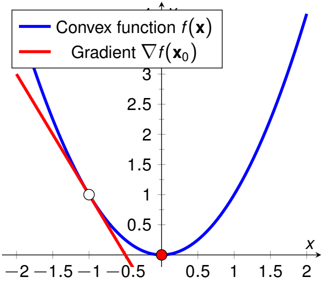
Does initialization matter?



Does initialization matter?

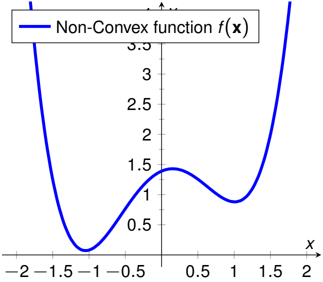


Does initialization matter?



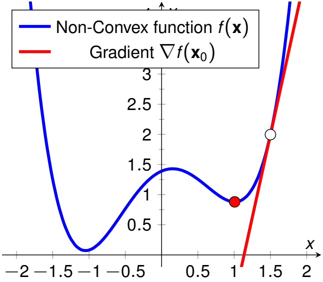
- No it doesn't for **convex** optimization problems

Does initialization matter?



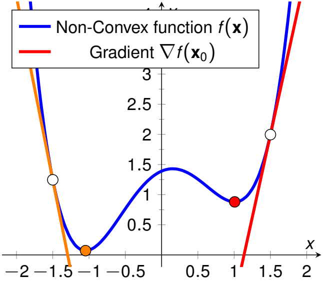
- No it doesn't for **convex** optimization problems

Does initialization matter?



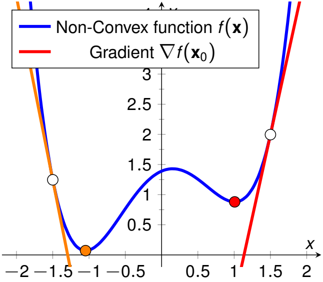
- No it doesn't for **convex** optimization problems

Does initialization matter?



- No it doesn't for **convex** optimization problems
- But it **does** for every **non-convex** problem

Does initialization matter?



- No it doesn't for **convex** optimization problems
- But it **does** for every **non-convex** problem
- Neural Networks with a non-linearity are in general **non-convex**

Simple Initialization

Bias

- **Bias** units can simply be **initialized to zero**

Weights

Simple Initialization

Bias

- **Bias** units can simply be **initialized to zero**
- Using **ReLUs** a **small positive** constant (0.1) is better because of the **dying ReLU issue**

Weights

Simple Initialization

Bias

- **Bias** units can simply be **initialized to zero**
- Using **ReLUs** a **small positive** constant (0.1) is better because of the **dying ReLU issue**

Weights

- **Weights** need to be **random** to **break symmetry**
- It's **especially bad** to **initialize** them **with zeros** because the gradient is zero!

Simple Initialization

Bias

- **Bias** units can simply be **initialized to zero**
- Using **ReLUs** a **small positive** constant (0.1) is better because of the **dying ReLU issue**

Weights

- **Weights** need to be **random** to **break symmetry**
- It's **especially bad** to **initialize** them **with zeros** because the gradient is zero!
- Similar to the learning rate their **variance** influences the **stability** of learning
- **Small** uniform/Gaussian values work

Calibrating the variances

- Suppose we have a single **linear** neuron with weights **W** and an input **X**
- Capital letters here mark them as **random variables**!

Calibrating the variances

- Suppose we have a single **linear** neuron with weights **W** and an input **X**
- Capital letters here mark them as **random variables**!
- Output is:

$$\hat{Y} = \mathbf{WX} = \left(\sum_{n=1}^{N} W_n X_n \right) + b$$

Calibrating the variances

- Suppose we have a single **linear** neuron with weights **W** and an input **X**
- Capital letters here mark them as **random variables**!
- Output is:

$$\hat{Y} = \mathbf{W}\mathbf{X} = \left(\sum_{n=1}^{N} W_n X_n\right) + b$$

- We are interested in $\mathrm{Var}(\hat{Y})$

Calibrating the variances

- Suppose we have a single **linear** neuron with weights **W** and an input **X**
- Capital letters here mark them as **random variables**!
- Output is:

$$\hat{Y} = \mathbf{W}\mathbf{X} = \left(\sum_{n=1}^{N} W_n X_n\right) + b$$

- We are interested in $\mathrm{Var}(\hat{Y})$
- If we assume the W_n and X_n independent, the variance of every product is:

$$\mathrm{Var}(W_n X_n) = \mathbb{E}[X_n]^2 \mathrm{Var}(W_n) + \mathbb{E}[W_n]^2 \mathrm{Var}(X_n) + \mathrm{Var}(W_n)\mathrm{Var}(X_n)$$

Calibrating the variances

- Suppose we have a single **linear** neuron with weights **W** and an input **X**
- Capital letters here mark them as **random variables**!
- Output is:

$$\hat{Y} = \mathbf{WX} = \left(\sum_{n=1}^{N} W_n X_n \right) + b$$

- We are interested in $\mathrm{Var}(\hat{Y})$
- If we assume the W_n and X_n independent, the variance of every product is:

$$\mathrm{Var}(W_n X_n) = \mathbb{E}[X_n]^2 \mathrm{Var}(W_n) + \mathbb{E}[W_n]^2 \mathrm{Var}(X_n) + \mathrm{Var}(W_n)\mathrm{Var}(X_n)$$

- If W_n and X_n have zero-mean: $\mathrm{Var}(W_n X_n) = \mathrm{Var}(W_n)\mathrm{Var}(X_n)$

Calibrating the variances

- Suppose we have a single **linear** neuron with weights **W** and an input **X**
- Capital letters here mark them as **random variables**!
- Output is:

$$\hat{Y} = \mathbf{WX} = \left(\sum_{n=1}^{N} W_n X_n\right) + b$$

- We are interested in $\mathrm{Var}(\hat{Y})$
- If we assume the W_n and X_n independent, the variance of every product is:

$$\mathrm{Var}(W_n X_n) = \mathbb{E}[X_n]^2 \mathrm{Var}(W_n) + \mathbb{E}[W_n]^2 \mathrm{Var}(X_n) + \mathrm{Var}(W_n)\mathrm{Var}(X_n)$$

- If W_n and X_n have zero-mean: $\mathrm{Var}(W_n X_n) = \mathrm{Var}(W_n)\mathrm{Var}(X_n)$
- Now we assume the X_n and W_n to be **I**ndependent and **I**dentically **D**istributed:

$$\mathrm{Var}(\hat{Y}) = \underbrace{N\,\mathrm{Var}(W_n)}_{scales Var}\,\mathrm{Var}(X_n)$$

Xavier initialization

- We can "calibrate" the variances for the forward-pass

  by initializing with a zero-mean Gaussian: $\mathcal{N}(0, \sigma)$

  with $\sigma^2 = \frac{1}{\text{fan\_in}}$

  where "fan_in" is the **input** dimension of the weights

Xavier initialization

- We can "calibrate" the variances for the forward-pass

  by initializing with a zero-mean Gaussian: $\mathcal{N}(0, \sigma)$
  with $\sigma^2 = \frac{1}{\text{fan\_in}}$
  where "fan_in" is the **input** dimension of the weights

- However the backward-pass

  needs $\sigma^2 = \frac{1}{\text{fan\_out}}$
  where "fan_out" is the **output** dimension of the weights

Xavier initialization

- We can "calibrate" the variances for the forward-pass

  by initializing with a zero-mean Gaussian: $\mathcal{N}(0, \sigma)$

  with $\sigma^2 = \frac{1}{\text{fan\_in}}$

  where "fan_in" is the **input** dimension of the weights
- However the backward-pass

  needs $\sigma^2 = \frac{1}{\text{fan\_out}}$

  where "fan_out" is the **output** dimension of the weights
- So we average those two:

  $$\sigma = \sqrt{\frac{2}{\text{fan\_out} + \text{fan\_in}}}$$
- Named "**Xavier**" **initialization** after the first author [21]

He initialization

➔ the assumption of **linear** neurons is a problem

- He et al. [12] showed, for ReLUs it's better to use:

$$\sigma = \sqrt{\frac{2}{\text{fan\_in}}}$$

Conventional Initial Choices

- L_2 regularization
- Dropout using $p = 0.5$ for FCN, use selectively in CNNs
- Mean subtraction
- Batch normalization
- He initialization

FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Transfer Learning

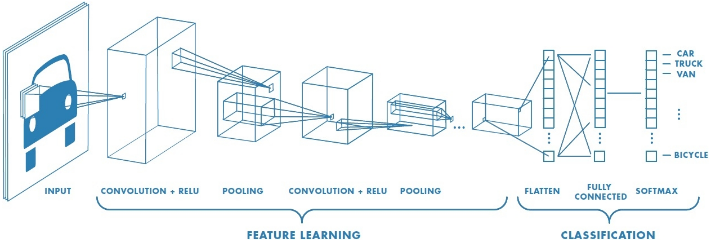
Transfer Learning

- Most **medical datasets** are prohibitively **small**

Transfer Learning

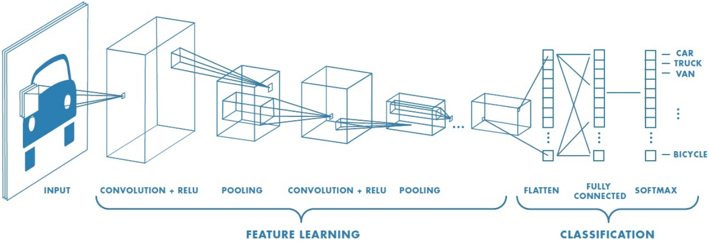
- Most **medical datasets** are prohibitively **small**
- **Reuse models** trained on e.g. ImageNet
  - → for a **different task** on the **same data**
  - → on **different data** for the **same task**
  - → on **different data** for a **different task**

Weight Transfer



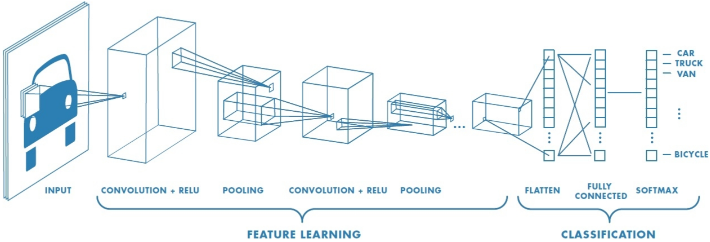
- What should we transfer?

Weight Transfer



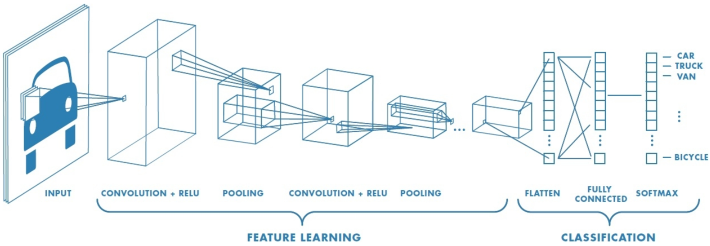
- What should we transfer?
  - ➔ Convolutional layers extract features

Weight Transfer



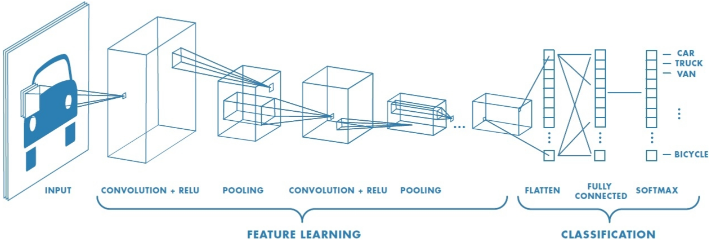
- What should we transfer?
  - ➔ Convolutional layers extract features
  - ➔ Expectation: Less task-specific in earlier layers

Weight Transfer



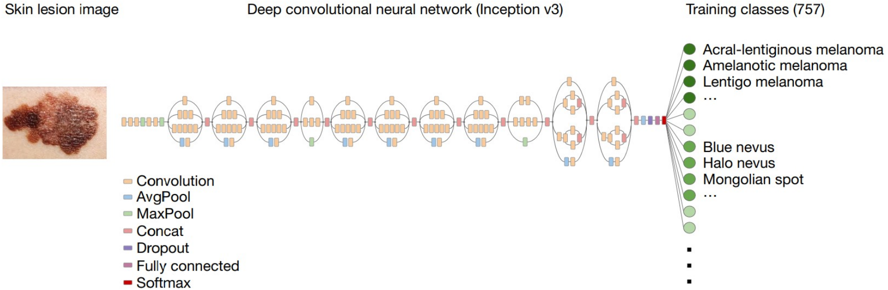
- What should we transfer?
  - ➔ Convolutional layers extract features
  - ➔ Expectation: Less task-specific in earlier layers
- We cut the network at some depth in the feature extraction part

Weight Transfer



- What should we transfer?
  - ➔ Convolutional layers extract features
  - ➔ Expectation: Less task-specific in earlier layers
- We cut the network at some depth in the feature extraction part
- The extracted parts can be
  1. fixed by setting $\eta = 0$
  2. **fine-tuned**

Example: Skin Cancer classification



- State-of-the-art architecture, pre-trained on ImageNet
- Fine-tuned on skin cancer data [5]

Source: https://www.nature.com/articles/nature21056

Transfer between modalities

- Also found to be **beneficial** if we transfer from RGB images to X-ray

Transfer between modalities

- Also found to be **beneficial** if we transfer from RGB images to X-ray

➔ It's sufficient to simply copy the input three times

- Finetuning usually necessary

Transfer between modalities

- Also found to be **beneficial** if we transfer from RGB images to X-ray

➔ It's sufficient to simply copy the input three times

- Finetuning usually necessary
- Alternative: Use feature representations of other network as a loss function
  ➔ Perceptual loss

Transfer between modalities

- Also found to be **beneficial** if we transfer from RGB images to X-ray

➔ It's sufficient to simply copy the input three times

- Finetuning usually necessary
- Alternative: Use feature representations of other network as a loss function
  ➔ Perceptual loss

➔ Always use transfer learning!

NEXT TIME

ON DEEP LEARNING

Regularization - Part 5

A. Maier, V. Christlein, K. Breininger, Z. Yang, L. Rist, M. Nau, S. Jaganathan, C. Liu, N. Maul, L. Folle, K. Packhäuser, M. Zinnen
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
April 24, 2023

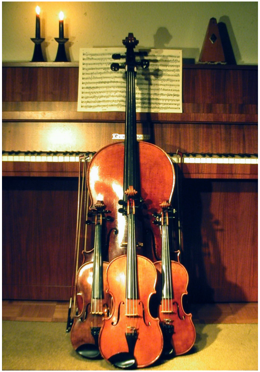
Multi-Task Learning (MTL)

Concept

- So far: One network for one task
- Transfer learning: **Reuse** a network
- Can we do more?

Concept

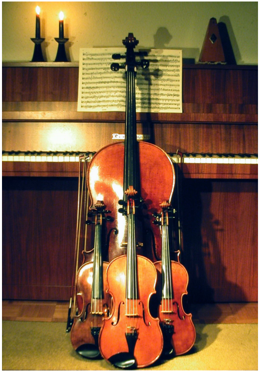
- So far: One network for one task
- Transfer learning: **Reuse** a network
- Can we do more?
- Real world examples:
  - Learning to play the piano and the violin
    ➔ Both require good hearing, sense of rhythm, music notation, . . .



Source: https://commons.wikimedia.org/wiki/File:Klavierquintett.JPG

Concept

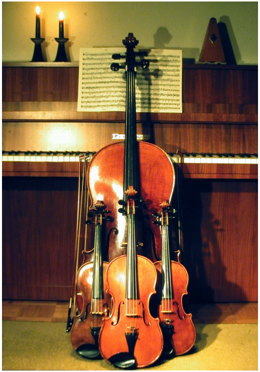
- So far: One network for one task
- Transfer learning: **Reuse** a network
- Can we do more?
- Real world examples:
  - Learning to play the piano and the violin
    ➔ Both require good hearing, sense of rhythm, music notation, . . .
  - Soccer and basketball training
    ➔ Both require stamina, speed, body awareness, body-eye coordination, . . .



Source: https://commons.wikimedia.org/wiki/File:Klavierquintett.JPG

Concept

- So far: One network for one task
- Transfer learning: **Reuse** a network
- Can we do more?
- Real world examples:
  - Learning to play the piano and the violin
    ➔ Both require good hearing, sense of rhythm, music notation, . . .
  - Soccer and basketball training
    ➔ Both require stamina, speed, body awareness, body-eye coordination, . . .



Even better than reusing: Learning them **simultaneously** can provide a better understanding of the **shared underlying concepts**.

Source: https://commons.wikimedia.org/wiki/File:Klavierquintett.JPG

Concept (cont.)

- Idea: Train a network **simultaneously** on **multiple related** tasks

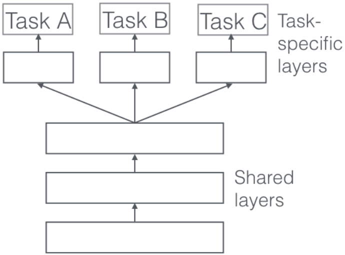
Concept (cont.)

- Idea: Train a network **simultaneously** on **multiple related** tasks
- Adapt loss function to assess performance for multiple tasks

Concept (cont.)

- Idea: Train a network **simultaneously** on **multiple related** tasks
- Adapt loss function to assess performance for multiple tasks
- Multi task learning introduces an **inductive bias**: We prefer a model that can explain more than one task
- Reduces risk of **overfitting** on one particular task [2]
  ➔ model generalizes better

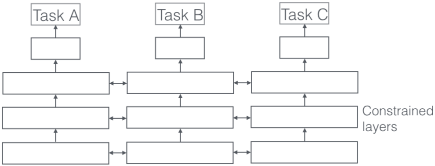
Hard parameter sharing



- Several hidden layers shared between all tasks
- Additional task-specific output layers
- Baxter '97 [2]: Multi-task learning of N tasks reduces chance of overfitting by an order of N

Source: http://ruder.io/multi-task/

Soft parameter sharing



- Each model has its own parameters
- Instead of forcing equality, distance between parameters is regularized as part of the loss function
- Options e.g. l_2-norm, trace-norm, ...

Source: http://ruder.io/multi-task/

Auxiliary tasks

- Additional tasks have own purpose or are just **auxiliary** to the original task

Auxiliary tasks

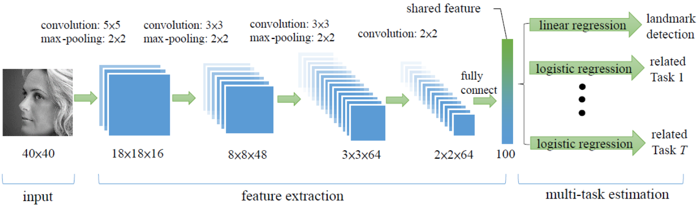
- Additional tasks have own purpose or are just **auxiliary** to the original task
- Example: **Facial Landmark Detection** by Zhang et al. 2014 [22]

Auxiliary tasks

- Additional tasks have own purpose or are just **auxiliary** to the original task
- Example: **Facial Landmark Detection** by Zhang et al. 2014 [22]
- Facial landmark detection impeded by occlusion and pose variances

Auxiliary tasks

- Additional tasks have own purpose or are just **auxiliary** to the original task
- Example: **Facial Landmark Detection** by Zhang et al. 2014 [22]
- Facial landmark detection impeded by occlusion and pose variances
- Simultaneously learn to estimate landmarks **and** "subtly" related task:
    - Face pose
    - Smiling/not smiling
    - Glasses/no glasses (occlusion)
    - Gender



Source: [22]

Auxiliary tasks (cont.)

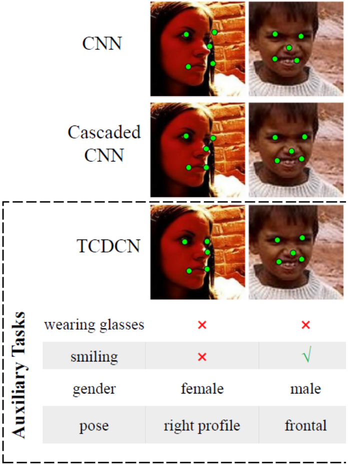
Auxiliary Tasks							
wearing glasses	×	×	√	×	√	×	×
smiling	×	√	×	×	×	×	×
gender	female	male	female	female	male	male	female
pose	right profile	frontal	frontal	left	frontal	frontal	right profile

Landmark detection

Source: [22]

Auxiliary tasks (cont.)

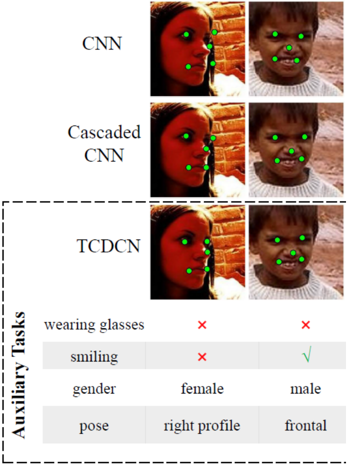
- Certain features difficult to learn for one task but easy for a related one [4]



Source: [22]

Auxiliary tasks (cont.)

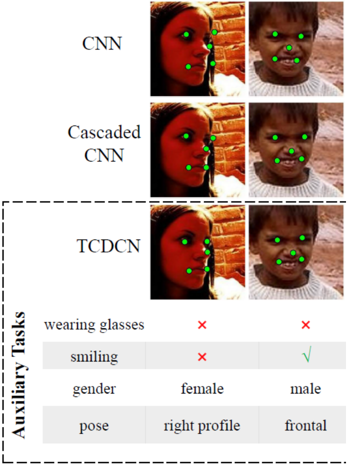
- Certain features difficult to learn for one task but easy for a related one [4]
- Auxiliary tasks can help to “steer” training in a specific direction
- Include prior knowledge by choosing appropriate auxiliary tasks



Source: [22]

Auxiliary tasks (cont.)

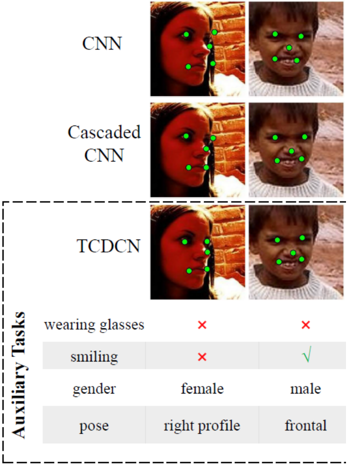
- Certain features difficult to learn for one task but easy for a related one [4]
- Auxiliary tasks can help to “steer” training in a specific direction
- Include prior knowledge by choosing appropriate auxiliary tasks
- Tasks can have different convergence rates
  ➔ Task-based early-stopping



Source: [22]

Auxiliary tasks (cont.)

- Certain features difficult to learn for one task but easy for a related one [4]
- Auxiliary tasks can help to “steer” training in a specific direction
- Include prior knowledge by choosing appropriate auxiliary tasks
- Tasks can have different convergence rates
  ➔ Task-based early-stopping
- Open research question: **What are appropriate auxiliary tasks?**



Source: [22]

NEXT TIME

ON DEEP LEARNING

Coming Up

- Practical recommendations to make training work
- Evaluation of performance
- Methods to deal with common problems
- Concrete case studies using all the pieces you now learned

Comprehensive Questions

- What is the bias-variance tradeoff?
- What is model capacity?
- Describe three techniques to address overfitting in a neural network.
- Why do we often need a validation set?
- Can we optimize the hyperparameter λ by gradient descend on the training set?
- What connects the covariate shift problem and the ReLU?
- How can we address the covariate shift problem?
- Which problem do current initialization schemes try to address?
- What is transfer learning?

Further Reading

- Link - for details on Maximum A Posteriori estimation and the bias-variance decomposition
- Link - for a comprehensive text about practical recommendations for regularization
- Link - the paper about calibrating the variances

References

References I

[1] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: Proceedings of The 32nd International Conference on Machine Learning. 2015, pp. 448–456.

[2] Jonathan Baxter. “A Bayesian/Information Theoretic Model of Learning to Learn via Multiple Task Sampling”. In: Machine Learning 28.1 (July 1997), pp. 7–39.

[3] Christopher M. Bishop. Pattern Recognition and Machine Learning (Information Science and Statistics) Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

References II

[4] Richard Caruana. “Multitask Learning: A Knowledge-Based Source of Inductive Bias”. In: Proceedings of the Tenth International Conference on Machine Learning. Morgan Kaufmann, 1993, pp. 41–48.

[5] Andre Esteva, Brett Kuprel, Roberto A Novoa, et al. “Dermatologist-level classification of skin cancer with deep neural networks”. In: Nature 542.7639 (2017), pp. 115–118.

[6] C. Ding, C. Xu, and D. Tao. “Multi-Task Pose-Invariant Face Recognition”. In: IEEE Transactions on Image Processing 24.3 (Mar. 2015), pp. 980–993.

[7] Li Wan, Matthew Zeiler, Sixin Zhang, et al. “Regularization of neural networks using dropconnect”. In: Proceedings of the 30th International Conference on Machine Learning (ICML-2013, pp. 1058–1066.

References III

[8] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, et al. “Dropout: a simple way to prevent neural networks from overfitting.”. In: Journal of Machine Learning Research 15.1 (2014), pp. 1929–1958.

[9] R. O. Duda, P. E. Hart, and D. G. Stork. Pattern Classification. John Wiley and Sons, inc., 2000.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. http://www.deeplearningbook.org. MIT Press, 2016.

[11] Yuxin Wu and Kaiming He. “Group normalization”. In: arXiv preprint arXiv:1803.08494 (2018).

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, et al. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: Proceedings of the IEEE international conference on computer vision. 2015, pp. 1026–1034.

References IV

[13] D Ulyanov, A Vedaldi, and VS Lempitsky. Instance normalization: the missing ingredient for fast stylization. CoRR abs/16

[14] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, et al. “Self-Normalizing Neural Networks”. In: Advances in Neural Information Processing Systems (NIPS). Vol. abs/1706.02515. 2017. arXiv: 1706.02515.

[15] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. “Layer normalization”. In: arXiv preprint arXiv:1607.06450 (2016).

[16] Nima Tajbakhsh, Jae Y Shin, Suryakanth R Gurudu, et al. “Convolutional neural networks for medical image analysis: Full training or fine tuning?” In: IEEE transactions on medical imaging 35.5 (2016), pp. 1299–1312.

References V

[17] Yoshua Bengio. "Practical recommendations for gradient-based training of deep architectures". In: Neural networks: Tricks of the trade. Springer, 2012, pp. 437–478.

[18] Chiyuan Zhang, Samy Bengio, Moritz Hardt, et al. "Understanding deep learning requires rethinking generalization". In: arXiv preprint arXiv:1611.03530 (2016).

[19] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, et al. "How Does Batch Normalization Help Optimization?" In: arXiv e-prints, arXiv:1805.11604 (May 2018), arXiv:1805.11604. arXiv: 1805.11604 [stat.ML].

[20] Tim Salimans and Diederik P Kingma. "Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks". In: Advances in Neural Information Processing Systems 29. Curran Associates, Inc., 2016, pp. 901–909.

References VI

[21] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: Proceedings of the Thirteenth International Conference on Artificial Intelligence 2010, pp. 249–256.

[22] Zhanpeng Zhang, Ping Luo, Chen Change Loy, et al. "Facial Landmark Detection by Deep Multi-task Learning". In: Computer Vision – ECCV 2014: 13th European Conference, Zurich, Switzerlan Cham: Springer International Publishing, 2014, pp. 94–108.