Deep Learning & Applied AI

Regularization, batchnorm and dropout

Emanuele Rodolà
rodola@di.uniroma1.it

SAPIENZA
UNIVERSITÀ DI ROMA

2nd semester a.y. 2024/2025 · April 14, 2025

Regularization

Any modification that is intended to reduce the generalization error but not the training error.

Goodfellow et al., "Deep Learning", MIT Press 2016

Weight penalties

$$\underbrace{\ell(\mathbf{\Theta})}_{\text{loss}} \;+\; \lambda \; \underbrace{\rho(\mathbf{\Theta})}_{\text{regularizer}}$$

Weight penalties

$$\underbrace{\ell(\boldsymbol{\Theta})}_{\text{loss}} \;+\; \lambda\; \underbrace{\rho(\boldsymbol{\Theta})}_{\text{regularizer}}$$

The regularizer induces a trade-off:

data fidelity vs. model complexity

Weight penalties

$$\underbrace{\ell(\mathbf{\Theta})}_{\text{loss}} \; + \; \lambda \; \underbrace{\rho(\mathbf{\Theta})}_{\text{regularizer}}$$

The regularizer induces a trade-off:

data fidelity vs. model complexity

Typical penalties:

- Tikhonov (L_2) regularization $\Rightarrow$ promotes shrinkage

Weight penalties

$$\underbrace{\ell(\mathbf{\Theta})}_{\text{loss}} + \lambda \underbrace{\rho(\mathbf{\Theta})}_{\text{regularizer}}$$

The regularizer induces a trade-off:

data fidelity vs. model complexity

Typical penalties:

- Tikhonov (L_2) regularization $\Rightarrow$ promotes shrinkage
- Lasso (L_1) $\Rightarrow$ promotes sparsity or weight selection

Weight penalties

$$\underbrace{\ell(\boldsymbol{\Theta})}_{\text{loss}} + \lambda \underbrace{\rho(\boldsymbol{\Theta})}_{\text{regularizer}}$$

The regularizer induces a trade-off:

data fidelity vs. model complexity

Typical penalties:

- Tikhonov (L_2) regularization $\Rightarrow$ promotes shrinkage
- Lasso (L_1) $\Rightarrow$ promotes sparsity or weight selection
- Bounded L_2 norm at each layer $\|\mathbf{W}^{(\ell)}\|_F \leq c^{(\ell)}$

Weight penalties

$$\underbrace{\ell(\boldsymbol{\Theta})}_{\text{loss}} \;+\; \lambda \underbrace{\rho(\boldsymbol{\Theta})}_{\text{regularizer}}$$

The regularizer induces a trade-off:

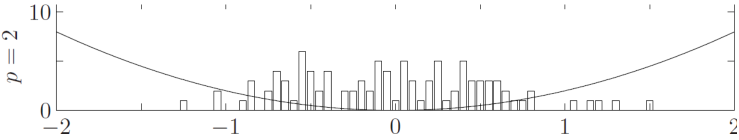
data fidelity vs. model complexity

Typical penalties:

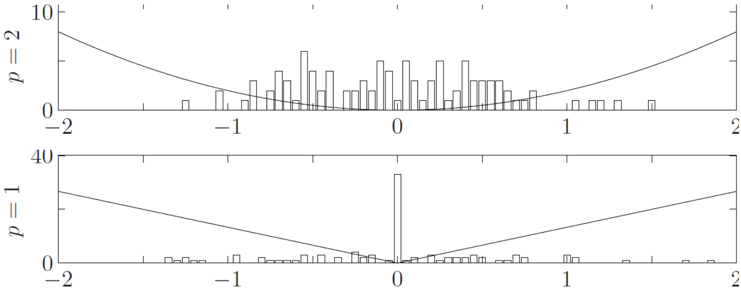
- Tikhonov (L_2) regularization $\Rightarrow$ promotes shrinkage
- Lasso (L_1) $\Rightarrow$ promotes sparsity or weight selection
- Bounded L_2 norm at each layer $\|\mathbf{W}^{(\ell)}\|_F \leq c^{(\ell)}$

After training, the L_p magnitude of each weight reflects its importance.

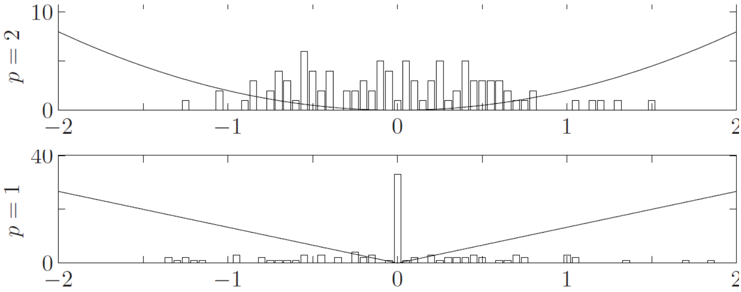
L_2 vs L_1 penalties



L_2 vs L_1 penalties

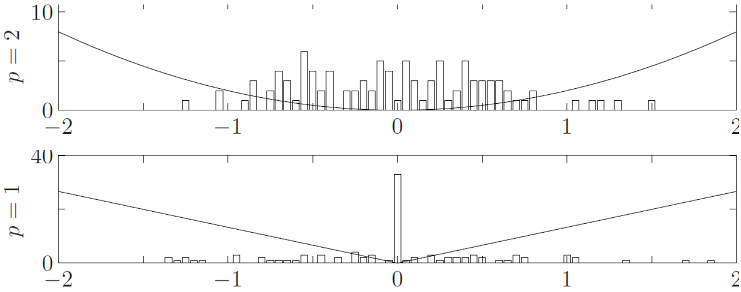


L_2 vs L_1 penalties



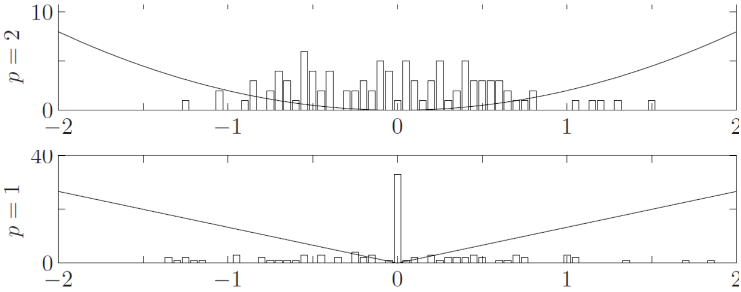
- Big reduction in $\|\boldsymbol{\Theta}\|_2$ if you scale down the values > 1

L_2 vs L_1 penalties



- Big reduction in $\|\mathbf{\Theta}\|_2$ if you scale down the values > 1
- Almost no reduction in $\|\mathbf{\Theta}\|_2$ for values < 1. Sparsity is discouraged!

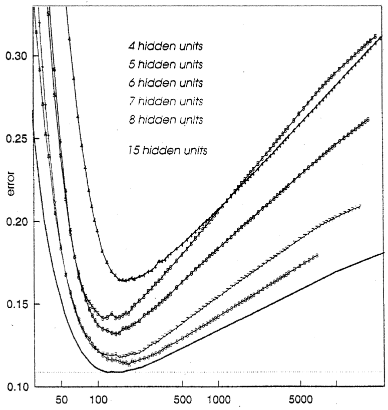
L_2 vs L_1 penalties



- Big reduction in $\|\mathbf{\Theta}\|_2$ if you scale down the values > 1
- Almost no reduction in $\|\mathbf{\Theta}\|_2$ for values < 1. Sparsity is discouraged!
- All the values are treated the same in $\|\mathbf{\Theta}\|_1$, no matter if they are > 1 or < 1. Any value can be set to zero, leading to sparse solutions.

Detecting overfitting

Overfitting can be recognized by looking at the validation error:

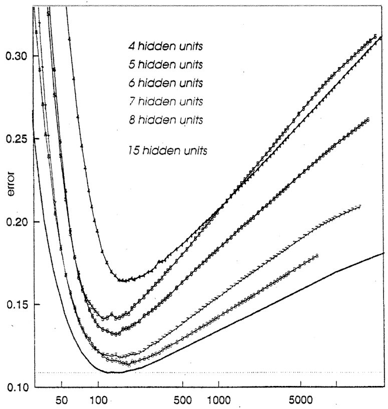


Weigend, "On overfitting and the effective number of hidden units", 1993

Detecting overfitting

Overfitting can be recognized by looking at the validation error:

- Small networks can also overfit.

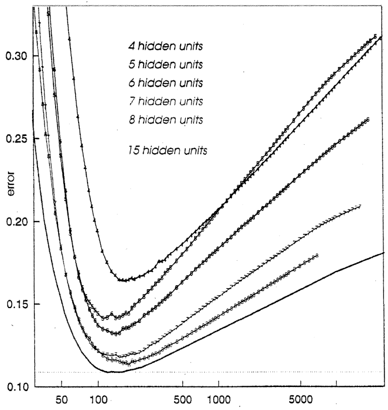


Weigend, "On overfitting and the effective number of hidden units", 1993

Detecting overfitting

Overfitting can be recognized by looking at the validation error:

- Small networks can also overfit.
- Large networks have best performance if they stop early.

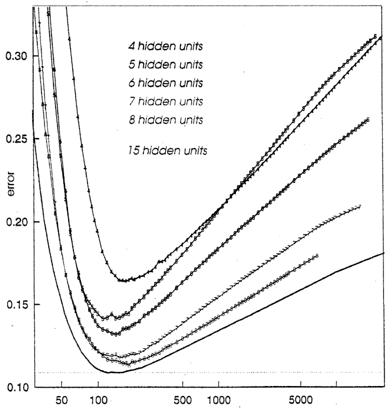


Weigend, "On overfitting and the effective number of hidden units", 1993

Detecting overfitting: Early stopping

Overfitting can be recognized by looking at the validation error:

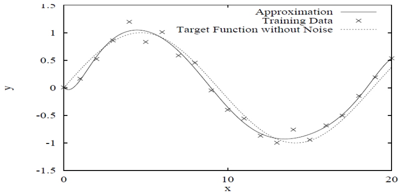
- Small networks can also overfit.
- Large networks have best performance if they stop early.
- Early stopping: Stop training as soon as performance on a validation set decreases.



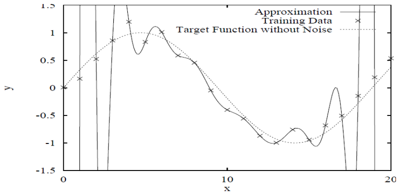
Weigend, "On overfitting and the effective number of hidden units", 1993

Many parameters $\neq$ overfitting

Typical overfitting with polynomial regression:



Order 10

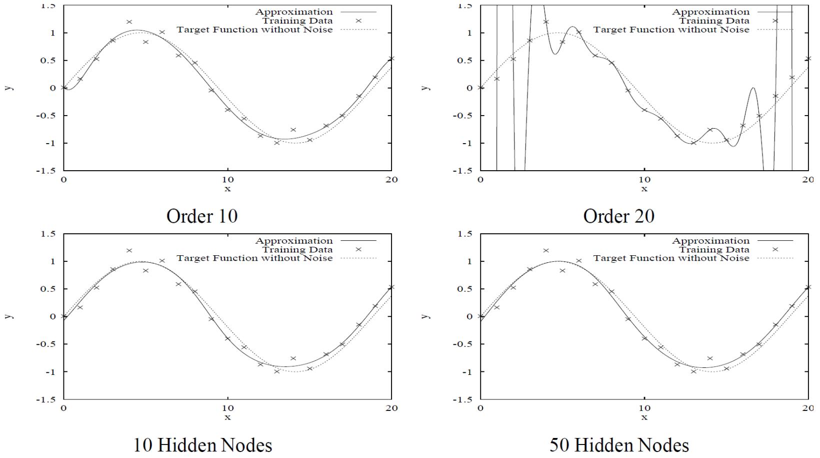


Order 20

Caruana et al, "Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping", NIPS 2001

Many parameters ≠ overfitting

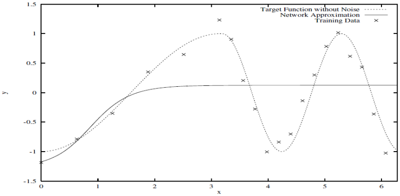
...but more MLP parameters not always lead to overfitting:



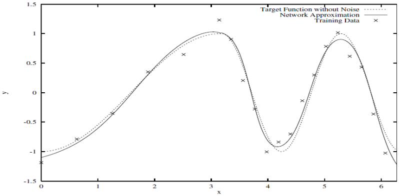
Caruana et al, "Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping", NIPS 2001

Many parameters ≠ overfitting

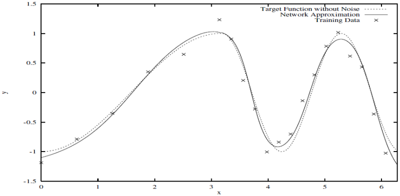
Good fit over all the different data regions:



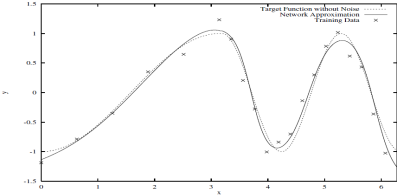
1 Hidden Unit



4 Hidden Units



10 Hidden Units

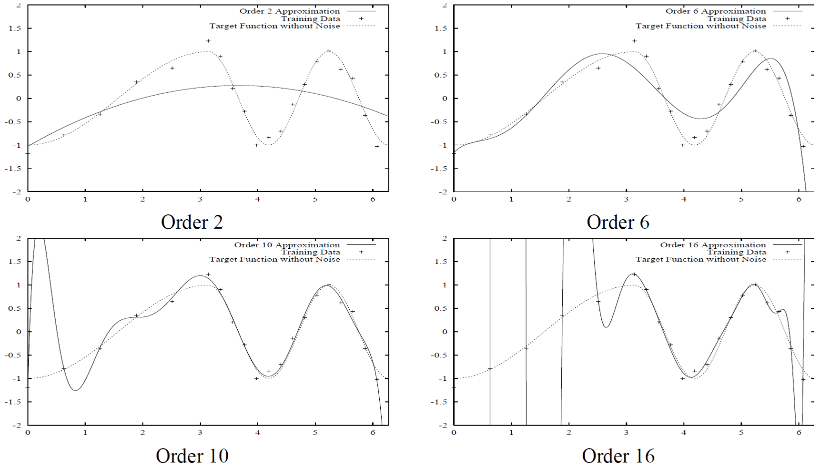


100 Hidden Units

Caruana et al, "Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping", NIPS 2001

Overfitting as a local phenomenon

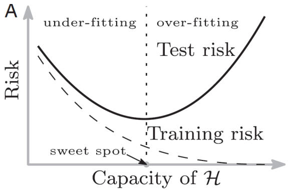
Overfitting is local and can vary significantly in different regions:



Caruana et al, "Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping", NIPS 2001

Double descent

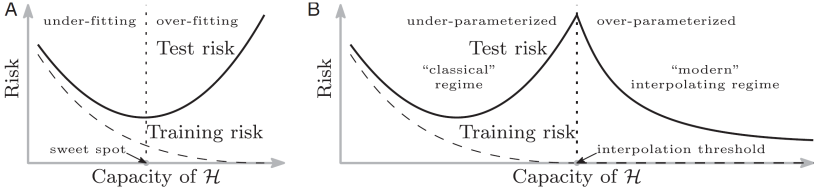
U-shaped curve as a function of # network parameters:



Belkin et al, "Reconciling modern machine-learning practice and the classical bias-variance trade-off", PNAS 2019

Double descent

U-shaped curve as a function of # network parameters:

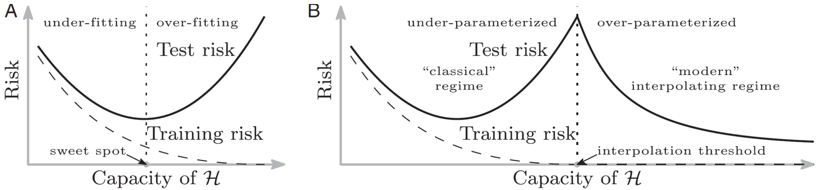


Interpolation: perfect fit on the training data.

Belkin et al, "Reconciling modern machine-learning practice and the classical bias-variance trade-off", PNAS 2019

Double descent

U-shaped curve as a function of # network parameters:



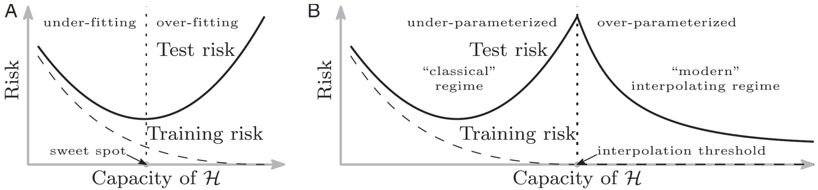
Interpolation: perfect fit on the training data.

"By considering larger function classes, which contain more candidate predictors compatible with the data, we are able to find interpolating functions that have smaller norm and are thus "simpler." Thus, increasing function class capacity improves performance of classifiers."

Belkin et al, "Reconciling modern machine-learning practice and the classical bias-variance trade-off", PNAS 2019

Double descent

U-shaped curve as a function of # network parameters:



Interpolation: perfect fit on the training data.

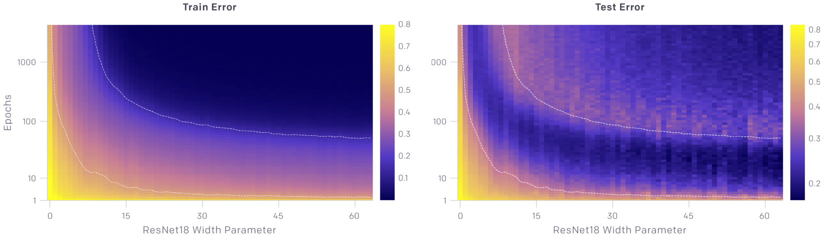
"By considering larger function classes, which contain more candidate predictors compatible with the data, we are able to find interpolating functions that have smaller norm and are thus "simpler." Thus, increasing function class capacity improves performance of classifiers."

What's surprising is that SGD finds such good models!

Belkin et al, "Reconciling modern machine-learning practice and the classical bias-variance trade-off", PNAS 2019

Epoch-wise double descent

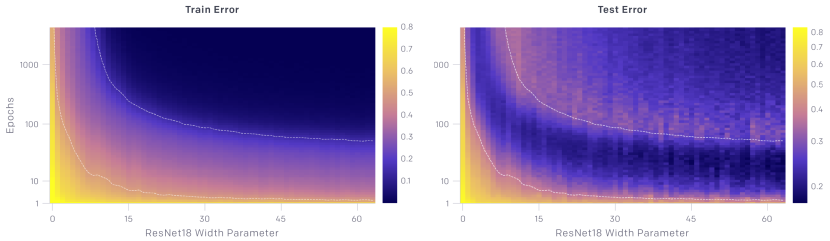
There is a regime where training longer reverses overfitting.



https://openai.com/blog/deep-double-descent/

Epoch-wise double descent

There is a regime where training longer reverses overfitting.

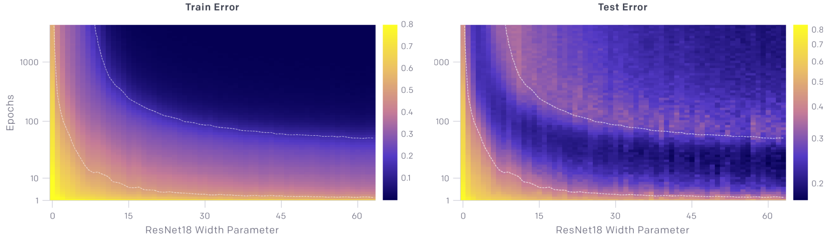


For a fixed number of epochs, the "usual" double descent.

https://openai.com/blog/deep-double-descent/

Epoch-wise double descent

There is a regime where training longer reverses overfitting.



For a fixed number of epochs, the "usual" double descent.

For a fixed number of parameters, we observe double descent as a function of training time.

https://openai.com/blog/deep-double-descent/

Batch normalization

$$\mathbf{x}^{(k)} = \sigma\left(\mathbf{W}^{(k)}\mathbf{x}^{(k-1)}\right)$$

Batch normalization

$$\mathbf{x}^{(k)} = \sigma\left(\mathbf{W}^{(k)}\mathbf{x}^{(k-1)}\right)$$

Internal covariate shift: The input distribution changes at each layer, and the layers need to continuously adapt to the new distribution.

Shimodaira, "Improving predictive inference under covariate shift by weighting the log-likelihood function", 2000

Batch normalization

$$\mathbf{x}^{(k)} = \sigma\left(\mathbf{W}^{(k)}\mathbf{x}^{(k-1)}\right)$$

Internal covariate shift: The input distribution changes at each layer, and the layers need to continuously adapt to the new distribution.

Normalize each feature map by the statistics computed within the mini-batch $\mathcal{X}$:

$$\hat{\mathbf{x}}^{(k)} = \text{normalize}(\mathbf{x}^{(k)}, \mathcal{X})$$

Shimodaira, "Improving predictive inference under covariate shift by weighting the log-likelihood function", 2000; Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

Batch normalization

$$\mathbf{x}^{(k)} = \sigma\left(\mathbf{W}^{(k)}\mathbf{x}^{(k-1)}\right)$$

Internal covariate shift: The input distribution changes at each layer, and the layers need to continuously adapt to the new distribution.

Normalize each feature map by the statistics computed within the mini-batch $\mathcal{X}$:

$$\hat{\mathbf{x}}^{(k)} = \text{normalize}(\mathbf{x}^{(k)}, \mathcal{X})$$

Note that both $\mathbf{x}$ and $\mathcal{X}$ depend on $\mathbf{W}$.

Backprop will then need the partial derivatives:

$$\frac{\partial}{\partial \mathbf{x}}\text{normalize}(\mathbf{x}, \mathcal{X}), \qquad \frac{\partial}{\partial \mathcal{X}}\text{normalize}(\mathbf{x}, \mathcal{X})$$

Shimodaira, "Improving predictive inference under covariate shift by weighting the log-likelihood function", 2000; Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

Batch normalization: Transformation

Transform each feature map $\mathbf{x}$ as:

$$\mathbf{x} \mapsto \frac{\mathbf{x} - \mathsf{E}[\mathcal{X}]}{\sigma(\mathcal{X})}$$

BN is per feature map, not per feature.

After the transformation, we get mean $= 0$ and std $= 1$.

Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

Batch normalization: Transformation

Transform each feature map $\mathbf{x}$ as:

$$\mathbf{x} \mapsto \frac{\mathbf{x} - \mathsf{E}[\mathcal{X}]}{\sigma(\mathcal{X})}$$

BN is per feature map, not per feature.

After the transformation, we get mean $= 0$ and std $= 1$.

Furthermore, introduce trainable weights:

$$\mathbf{x} \mapsto \gamma \frac{\mathbf{x} - \mathsf{E}[\mathcal{X}]}{\sigma(\mathcal{X})} + \beta$$

These allow to represent the identity $\mathbf{x} \mapsto \mathbf{x}$, if that is the optimal thing to do to solve the task.

Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

Batch normalization: Using mini-batches

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

Batch normalization: Using mini-batches

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

The batchnorm transformation makes each training example interact with the other examples in each mini-batch.

Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

Batch normalization: Properties

Typically applied right before the nonlinearity:

$$\sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \quad \text{becomes} \quad \sigma \circ \mathrm{BN}_{\gamma,\beta}(\mathbf{W}\mathbf{x})$$

The bias can be absorbed in the mean subtraction.

Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

Batch normalization: Properties

Typically applied right before the nonlinearity:

$$\sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \quad \text{becomes} \quad \sigma \circ \mathrm{BN}_{\gamma,\beta}(\mathbf{W}\mathbf{x})$$

The bias can be absorbed in the mean subtraction.

At test time, what μ and σ are used?

Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

Batch normalization: Properties

Typically applied right before the nonlinearity:

$$\sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \quad \text{becomes} \quad \sigma \circ \mathrm{BN}_{\gamma,\beta}(\mathbf{W}\mathbf{x})$$

The bias can be absorbed in the mean subtraction.

At test time, what μ and σ are used? BN tracks a running average during training, use those!

Ioffe and Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift”, ICML 2015

Batch normalization: Properties

Typically applied right before the nonlinearity:

$$\sigma(\mathbf{Wx} + \mathbf{b}) \quad \text{becomes} \quad \sigma \circ \mathrm{BN}_{\gamma,\beta}(\mathbf{Wx})$$

The bias can be absorbed in the mean subtraction.

At test time, what μ and σ are used? BN tracks a running average during training, use those!

Benefits:

- The stochastic uncertainty of the batch stats is a regularizer that can benefit generalization.

Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015; Santurkar et al, "How does batch normalization help optimization?", NIPS 2018

Batch normalization: Properties

Typically applied right before the nonlinearity:

$$\sigma(\mathbf{Wx} + \mathbf{b}) \quad \text{becomes} \quad \sigma \circ \mathrm{BN}_{\gamma,\beta}(\mathbf{Wx})$$

The bias can be absorbed in the mean subtraction.

At test time, what μ and σ are used? BN tracks a running average during training, use those!

Benefits:

- The stochastic uncertainty of the batch stats is a regularizer that can benefit generalization.
- BN leads to more stable gradients, thus faster training with higher learning rates.

Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015; Santurkar et al, "How does batch normalization help optimization?", NIPS 2018

Normalization variants

Normalizing along the batch dimension can lead to inconsistency:

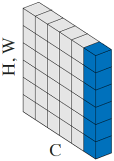
- Bad transfer across different data distributions.
- Reducing the mini-batch size increases error.

Normalization variants

Normalizing along the batch dimension can lead to inconsistency:

- Bad transfer across different data distributions.
- Reducing the mini-batch size increases error.

Several variants:

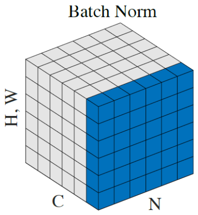


Normalization variants

Normalizing along the batch dimension can lead to inconsistency:

- Bad transfer across different data distributions.
- Reducing the mini-batch size increases error.

Several variants:

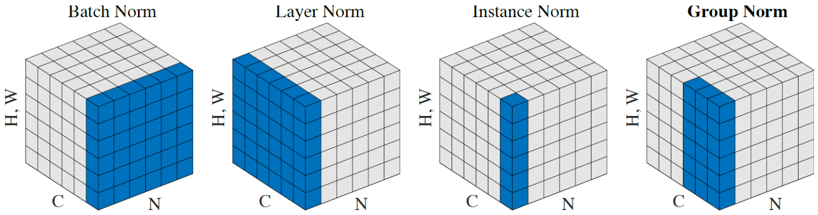


Normalization variants

Normalizing along the batch dimension can lead to inconsistency:

- Bad transfer across different data distributions.
- Reducing the mini-batch size increases error.

Several variants:



Wu and He, "Group normalization", ECCV 2018

Ensemble deep learning?

Assume you have lots of computational power.

Train an ensemble of deep nets and average their predictions.

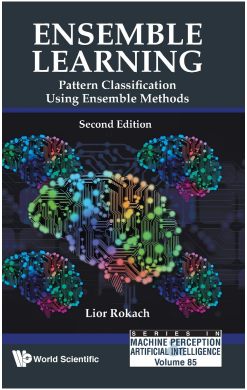
Ensemble deep learning?

Assume you have lots of computational power.

Train an ensemble of deep nets and average their predictions.

Ensembles (e.g. bayesian networks, random forests) tend to generalize better than the individual models!

Most successful methods in **Kaggle** are ensemble methods.



Neal, "Bayesian Learning for Neural Networks", 1996

Ensemble deep learning?

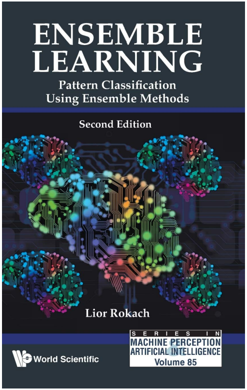
Assume you have lots of computational power.

Train an ensemble of deep nets and average their predictions.

Ensembles (e.g. bayesian networks, random forests) tend to generalize better than the individual models!

Most successful methods in **Kaggle** are ensemble methods.

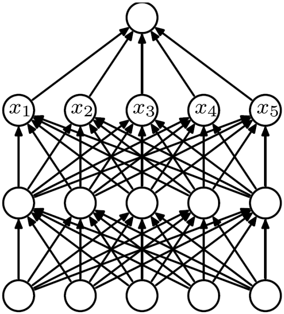
However, for deep nets this would come at a high computational cost.



Neal, "Bayesian Learning for Neural Networks", 1996

Dropout

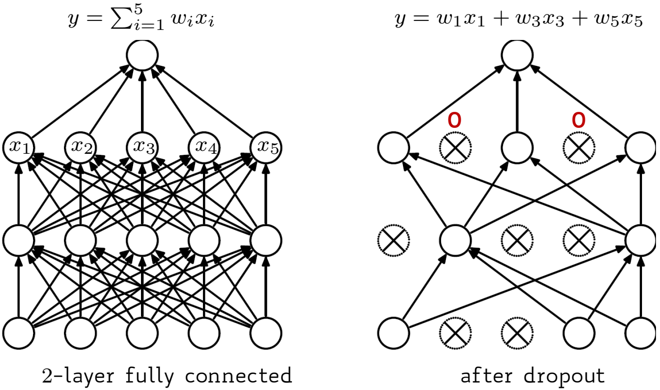
$y = \sum_{i=1}^{5} w_i x_i$



2-layer fully connected

Dropout

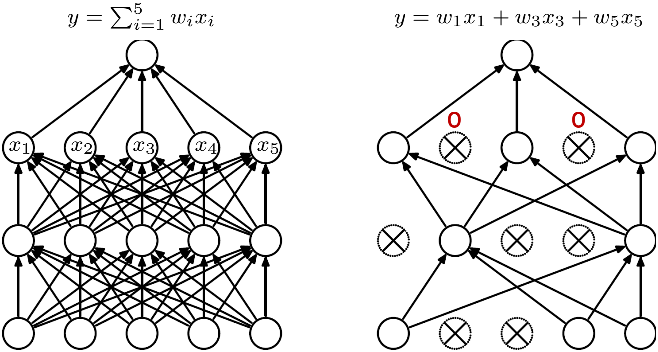
Main idea: Parametrize each model in the ensemble by dropping random units (i.e. nodes with their input/output connections):



Srivastava et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014

Dropout

Main idea: Parametrize each model in the ensemble by dropping random units (i.e. nodes with their input/output connections):



2-layer fully connected after dropout

Crucially, all networks share the same parameters.

Srivastava et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014

Dropout

Can be seen as sampling a network w.r.t. a probability distribution.

n nodes $\Rightarrow$ 2^n possible ways to sample them

Srivastava et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014

Dropout

Can be seen as sampling a network w.r.t. a probability distribution.

$$n \text{ nodes} \Rightarrow 2^n \text{ possible ways to sample them}$$

This is way too costly.

- **Training:** All the networks must be trained.
- **Test:** All the predictions must be averaged.

Srivastava et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014

Dropout

Can be seen as sampling a network w.r.t. a probability distribution.

$$n \text{ nodes} \Rightarrow 2^n \text{ possible ways to sample them}$$

Make it feasible by keeping one single network:

- **Training:** Generate a new sampling for each mini-batch in SGD.

Srivastava et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014

Dropout

Can be seen as sampling a network w.r.t. a probability distribution.

$$n \text{ nodes} \Rightarrow 2^n \text{ possible ways to sample them}$$

Make it feasible by keeping one single network:

- **Training:** Generate a new sampling for each mini-batch in SGD.

  The ensemble is trained to convergence.

  The individual models are not trained to convergence.

Srivastava et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014

Dropout

Can be seen as sampling a network w.r.t. a probability distribution.

$$n \text{ nodes} \Rightarrow 2^n \text{ possible ways to sample them}$$

Make it feasible by keeping one single network:

- **Test:** The trained weights from each model in the ensemble must be averaged somehow.

Srivastava et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014

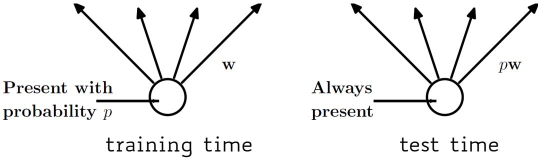
Dropout

Can be seen as sampling a network w.r.t. a probability distribution.

$$n \text{ nodes} \Rightarrow 2^n \text{ possible ways to sample them}$$

Make it feasible by keeping one single network:

- **Test:** The trained weights from each model in the ensemble must be averaged somehow.



If a unit is retained with probability p during training (chosen by hand for each layer), its outgoing weights are multiplied by p.

Srivastava et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014

Dropout as an ensemble method

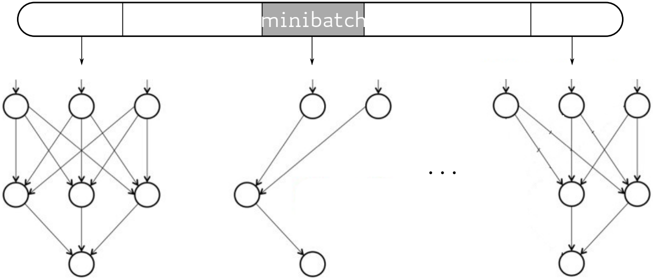
Dropout has two key features:

- Bagging, i.e. each model is trained on random data.
- Weight sharing, which is atypical in ensemble methods.

Dropout as an ensemble method

Dropout has two key features:

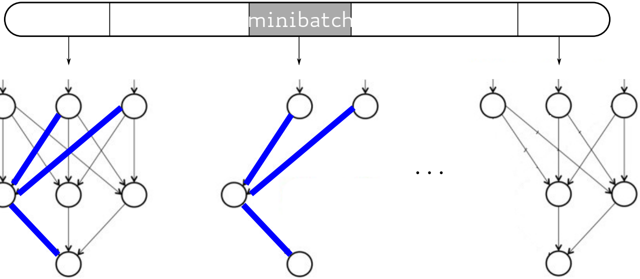
- Bagging, i.e. each model is trained on random data.
- Weight sharing, which is atypical in ensemble methods.



Dropout as an ensemble method

Dropout has two key features:

- Bagging, i.e. each model is trained on random data.
- Weight sharing, which is atypical in ensemble methods.



At each training step, the weight update is applied to all members of the ensemble simultaneously.

Dropout: Properties

In neural networks, weights are optimized jointly.

Co-adaptation: Small errors in a unit are absorbed by another unit.

Srivastava et al, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, JMLR 2014

Dropout: Properties

In neural networks, weights are optimized jointly.

Co-adaptation: Small errors in a unit are absorbed by another unit.

Some properties of dropout as a regularizer:

- Reduces co-adaptation by making units unreliable. This improves generalization to unseen data.

Srivastava et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014

Dropout: Properties

In neural networks, weights are optimized jointly.

Co-adaptation: Small errors in a unit are absorbed by another unit.

Some properties of dropout as a regularizer:

- Reduces co-adaptation by making units unreliable. This improves generalization to unseen data.
- Side-effect: sparse representations are learned.

Srivastava et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014

Dropout: Properties

In neural networks, weights are optimized jointly.

Co-adaptation: Small errors in a unit are absorbed by another unit.

Some properties of dropout as a regularizer:

- Reduces co-adaptation by making units unreliable. This improves generalization to unseen data.
- Side-effect: sparse representations are learned.
- Longer training times: parameter updates are now noisier.

Srivastava et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014; Warde-Farley et al, "An empirical analysis of dropout in piecewise linear networks", 2014

Dropout: Properties

In neural networks, weights are optimized jointly.

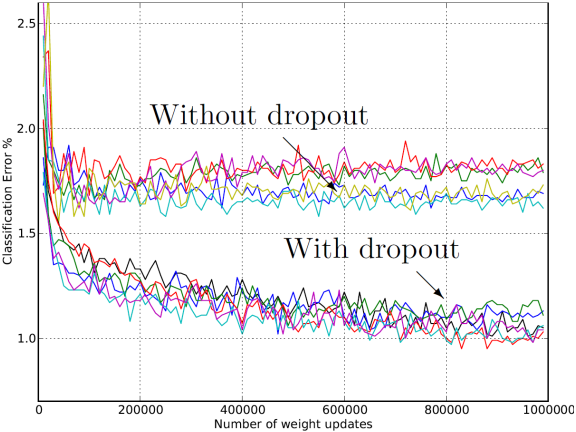
Co-adaptation: Small errors in a unit are absorbed by another unit.

Some properties of dropout as a regularizer:

- Reduces co-adaptation by making units unreliable. This improves generalization to unseen data.
- Side-effect: sparse representations are learned.
- Longer training times: parameter updates are now noisier.
- Typical choices: 20% of the input units and 50% of the hidden units.

Srivastava et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014; Warde-Farley et al, "An empirical analysis of dropout in piecewise linear networks", 2014

Dropout: Properties



Srivastava et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014

Suggested reading

- All the references given throughout the slides.
- Interesting thread on the history of double descent:
  https://twitter.com/hippopedoid/status/1243229021921579010
- *Section 4.2.1* is a practical guide for batchnorm by the original authors:
  https://arxiv.org/abs/1502.03167
- *Appendix A* is a practical guide for dropout by the original authors:
  http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf