Deep Learning

3. Regularization for Deep Learning

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute for Computer Science
University of Hildesheim, Germany

Syllabus

Tue. 21.4.	(1)	1. Supervised Learning (Review 1)
Tue. 28.4.	(2)	2. Neural Networks (Review 2)
Tue. 5.5.	(3)	3. Regularization for Deep Learning
Tue. 12.5.	(4)	4. Optimization for Training Deep Models
Tue. 19.5.	(5)	5. Convolutional Neural Networks
Tue. 26.5.	(6)	6. Recurrent Neural Networks
Tue. 2.6.	—	— *Pentecoste Break* —
Tue. 9.6.	(7)	7. Autoencoders
Tue. 16.6.	(8)	8. Generative Adversarial Networks
Tue. 23.6.	(9)	9. Recent Advances
Tue. 30.6.	(10)	10. Engineering Deep Learning Models
Tue. 7.7.	(11)	tbd.
Tue. 14.7.	(12)	Q & A

Outline

1. Overfitting and Underfitting
2. Parameter Shrinkage
3. Early Stopping
4. Dropout
5. Data Augmentation

Outline

1. Overfitting and Underfitting

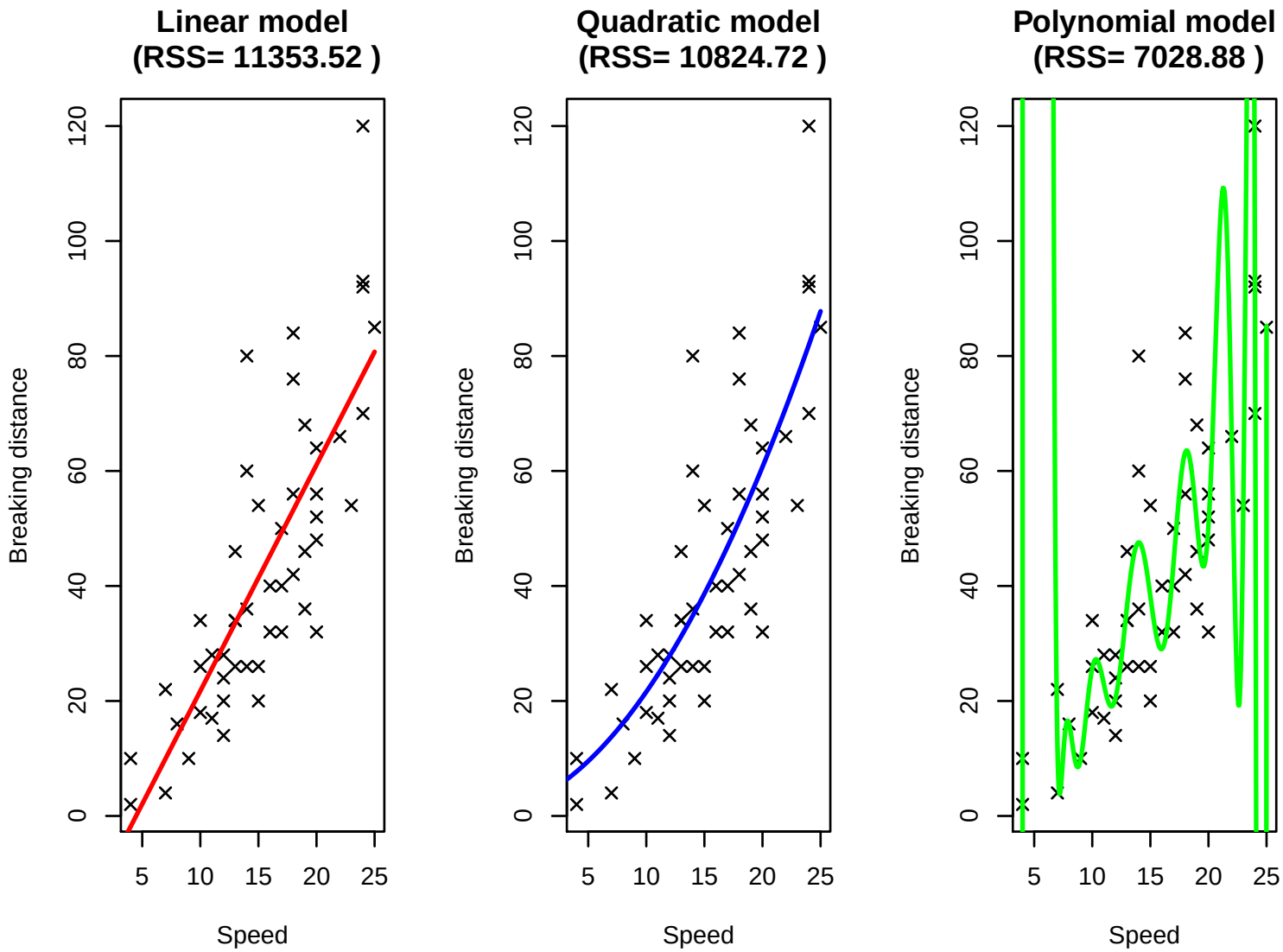
2. Parameter Shrinkage

3. Early Stopping

4. Dropout

5. Data Augmentation

Fitting of models



Underfitting/Overfitting

- **Underfitting:**
  - the model is not complex enough to explain the data well.
  - results in poor predictive performance.
- **Overfitting:**
  - the model is too complex, it describes the
    - noise, inherent random variations of the data generating process, instead of the
    - signal, the underlying relationship between target and predictors.
  - results in poor predictive performance as well.

Overfitting is Easy

- generally, overfitting is easy: given N points (x_n, y_n) without repeated measurements (i.e. $x_n \neq x_m, \ n \neq m$), there exists a polynomial of degree $N-1$ with RSS equal to 0.

$$\hat{y}(x) := \sum_{n=1}^{N} y_n \prod_{\substack{m=1 \\ m \neq n}}^{N} \frac{x - x_m}{x_n - x_m}$$

- neural networks can accomodate any number of parameters, using
  - wider networks: larger layer sizes / more neurons per layer and
  - deeper networks: more layers stacked on top of each other

  i.e., they can scale to arbitrary high capacities.

Regularization

- **regularization**: limit the capacity of a model to avoid overfitting
- **structural regularization**:
  use a model with limited number of parameters
  - i.e., a neural network with
    - limited width / layer size and
    - limited depth / number of layers
  - rule of thumb: one parameter for 10 data samples
    - very rough rule of thumb
    - if no further regularization technique is used
    - when also other regularization techniques are used,
      the rule is wrong by orders of magnitude!

Structural Regularization

- a neural network with one hidden layer is trained with varying layer sizes:

layer size	5	10	20	50	100
training loss	0.332	0.241	0.163	0.101	0.064
validation loss	0.355	0.278	0.290	0.296	0.301

- how wide is the hidden layer of the best regularized model?

Outline

1. Overfitting and Underfitting
2. Parameter Shrinkage
3. Early Stopping
4. Dropout
5. Data Augmentation

Parameter Shrinkage

- use only small parameter **values**
- add a penalty term to the objective function:

$$f(\theta; X, y) := \ell(\theta; X, y) + \lambda\Omega(\theta)$$

- $\lambda \in [0, \infty)$ called **regularization weight**
- regularize the neuron weights, but not the bias terms
- for simplicity use the same λ for all layers

Parameter Shrinkage / An Example

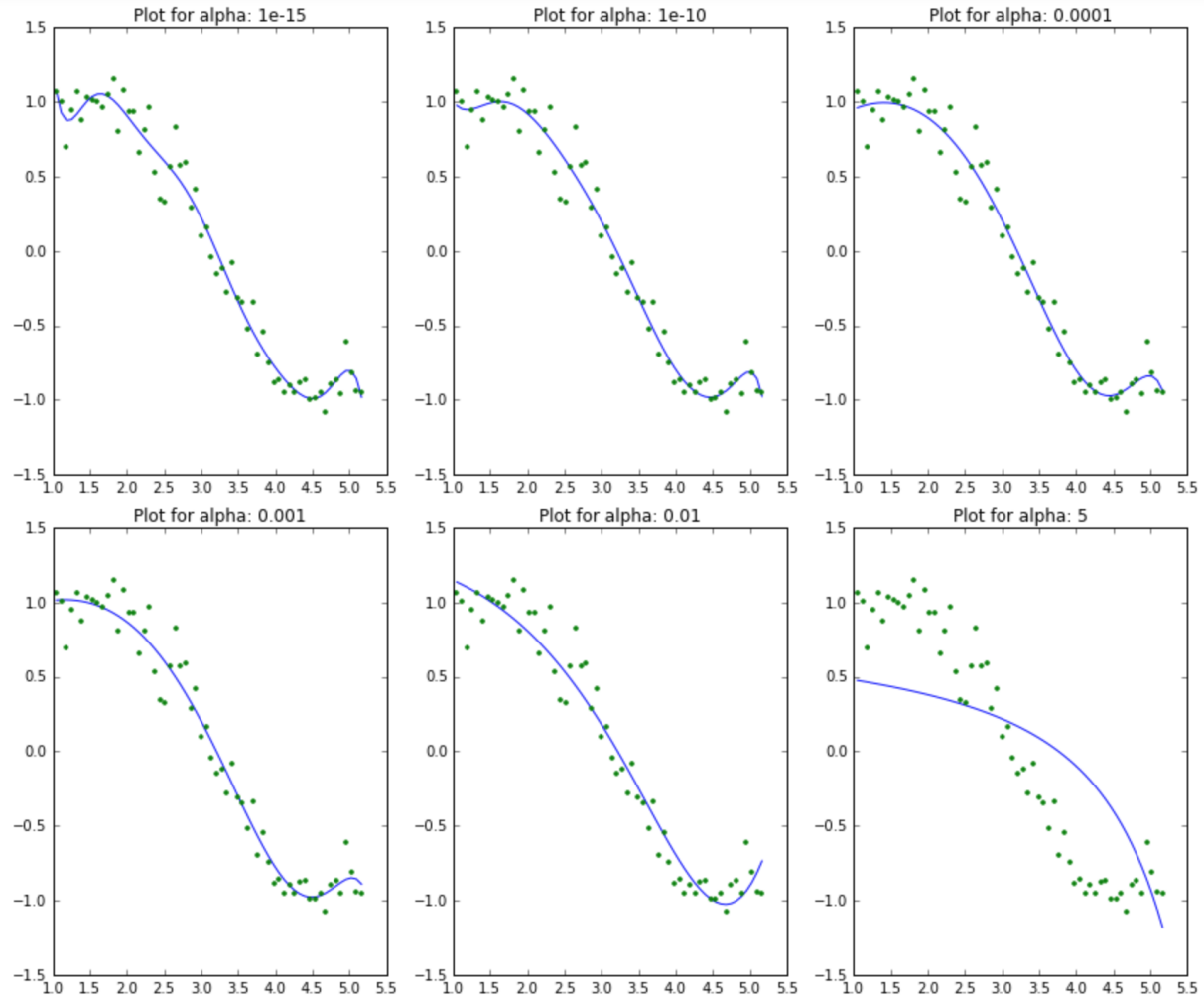


Figure 1: Regularizing polynomial regression (order 15), Source `www.analyticsvidhya.com`

L_2 Regularization

- L_2 regularization penalizes high θ values

$$f(\theta; X, y) = \ell(\theta; X, y) + \frac{\lambda}{2}\theta^T\theta$$

- gradients of the objective function:

$$\nabla_\theta f(\theta; X, y) = \nabla_\theta \ell(\theta; X, y) + \quad \ldots$$

- Q: What is the gradient of the L2 penality?
  A. $\lambda\theta$
  B. λ
  C. θ^2
  D. 0

L_2 Regularization

- L_2 regularization penalizes high θ values

$$f(\theta; X, y) = \ell(\theta; X, y) + \frac{\lambda}{2}\theta^T\theta$$

- gradients of the objective function:

$$\nabla_\theta f(\theta; X, y) = \nabla_\theta \ell(\theta; X, y) + \lambda\theta$$

- $\nabla_\theta \ell(\theta; X, y)$ can be computed through backpropagation
- a simple gradient descent step with a learning rate ϵ is:

$$\begin{aligned}\theta^{\text{new}} &:= \theta - \epsilon\left(\nabla_\theta \ell(\theta; X, y) + \lambda\theta\right) \\ &= (1 - \epsilon\lambda)\theta - \epsilon\nabla_\theta \ell(\theta; X, y)\end{aligned}$$

L_1 Regularization

- L_1 regularization:

$$f(\theta; X, y) := \ell(\theta; X, y) + \lambda ||\theta||_1$$
$$= \ell(\theta; X, y) + \lambda \sum_k |\theta_k|$$

- (sub)gradients of the objective function:

$$\nabla_\theta f(\theta; X, y) = \nabla_\theta \ell(\theta; X, y) + \lambda \left(\begin{cases} 1 & \text{if } \theta_k > 0 \\ -1 & \text{if } \theta_k \leq 0 \end{cases} \right)_{k=1:K}$$

- a simple gradient descent step with a learning rate ϵ is:

$$\theta^{\text{new}} := \theta - \epsilon \left(\nabla_\theta \ell(\theta; X, y) + \lambda (2\, \mathbb{I}(\theta > 0) - 1) \right)$$

Note: $\mathbb{I}(x) := 1$, if x is true, $:= 0$ otherwise. $\mathbb{I}(\theta > 0) = (\mathbb{I}(\theta_k > 0))_{k=1:K}$.

L1 and L2 Regularizations — Illustration of the Principle

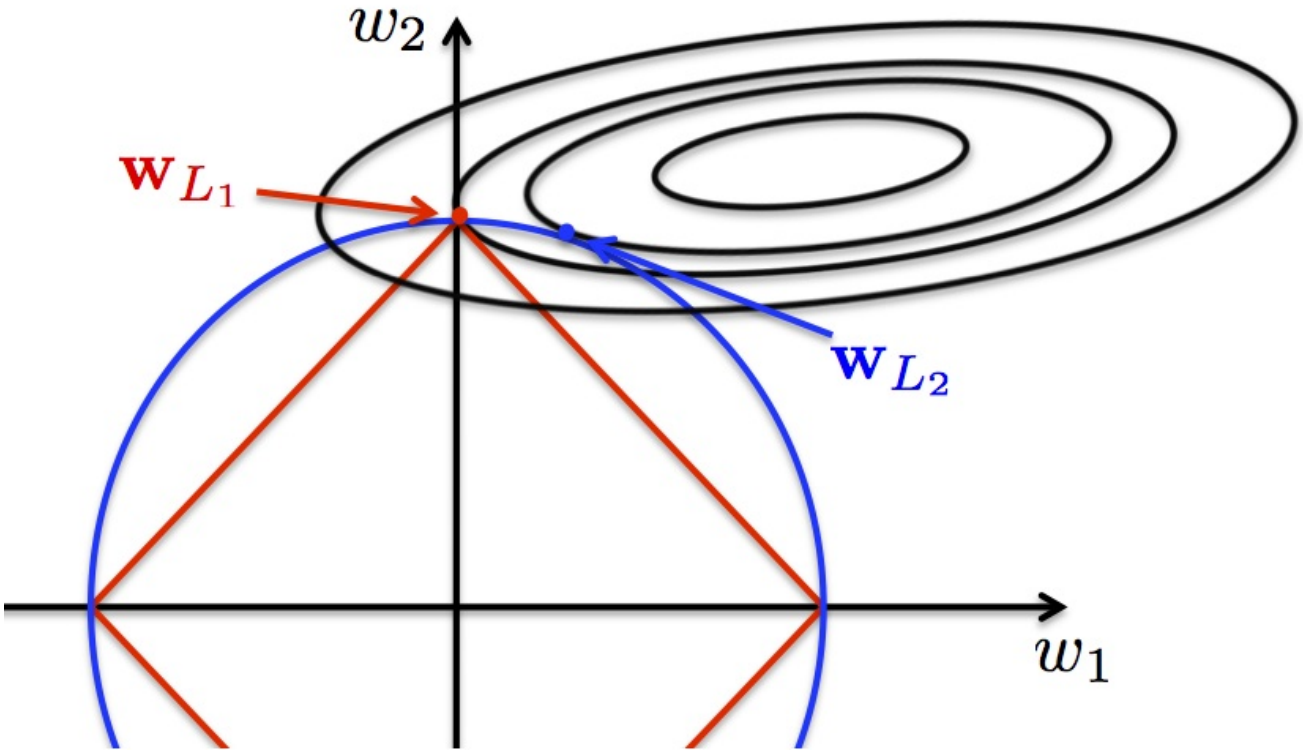


Figure 2: Competing objective terms. i) the blue line represents the L1 regularization, ii) the red line represents the L1 regularization, while iii) solid lines represent the objective function. Source: `g2pi.tsc.uc3m.es`

Constraining Parameters

- instead of encouraging small parameter values via minimizing an unconstrained, penalized objective:

$$\min f(\theta; X, y) := \ell(\theta; X, y) + \lambda\Omega(\theta), \quad \theta \in \mathbb{R}^K$$

- Hinton et al. 2012 propose to **constrain** parameter values:

$$\min f(\theta, \lambda; X, y) := \ell(\theta; X, y)$$
$$\text{w.r.t.} \quad |\theta_k| \leq \theta_{\text{max}}, \quad \forall k$$

with a given upper bound $\theta_{\text{max}} \in \mathbb{R}^+$ on parameter values

- project parameters back after updates:

$$\theta_k^{\text{proj}} := \min\{1, \frac{\theta_{\text{max}}}{|\theta_k|}\}\,\theta_k = \begin{cases} \frac{\theta_{\text{max}}}{|\theta_k|}\theta_k, & \text{if } |\theta_k| > \theta_{\text{max}} \\ \theta_k, & \text{else} \end{cases}$$

- less sensitive to learning rates,
  i.e., can be used with large initial learning rates

Outline

1. Overfitting and Underfitting

2. Parameter Shrinkage

3. Early Stopping

4. Dropout

5. Data Augmentation

Early Stopping / Motivation

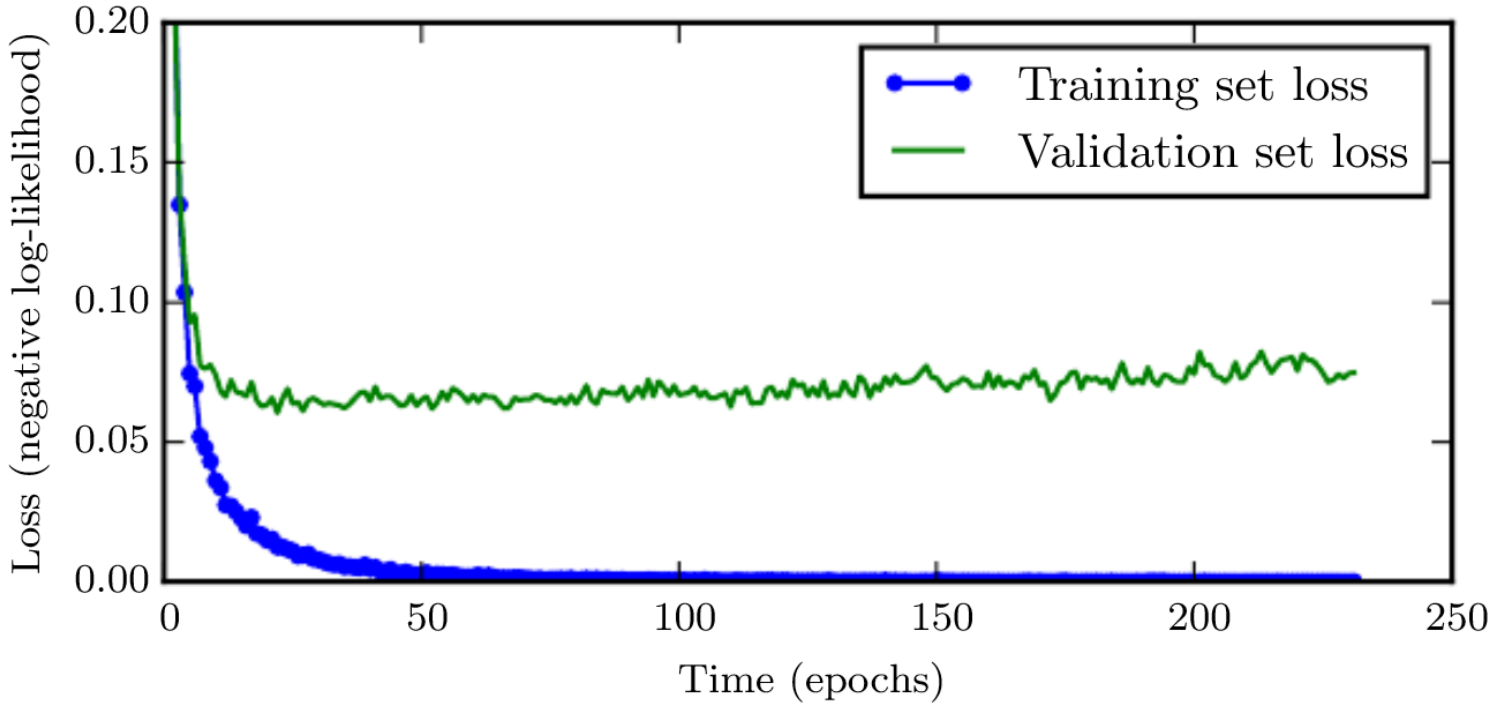


Figure 3: There is a better generalization in the earlier epochs of the optimization, Source: Goodfellow et al., 2016

Learning Parameters / Fixed Number of Iterations

min-fixed-iterations$(f, \mathcal{D}^{\text{train}}, \theta_0, I, \epsilon)$:

$\theta := \theta_0$

for $i \in (1, 2, 3, \ldots, I)$:

$\theta := \theta - \epsilon \nabla_\theta f(\theta; \mathcal{D}^{\text{train}})$

return θ

where

- $f : \mathbb{R}^K \times (\mathbb{R}^{M \times O})^* \to \mathbb{R}$ objective function
- $\mathcal{D}^{\text{train}} \in (\mathbb{R}^{M \times O})^*$ training dataset
- $\theta_0 \in \mathbb{R}^K$ initial parameter values (e.g., random)
- $I \in \mathbb{N}$ number of iterations
- $\epsilon \in \mathbb{R}^+$ learning rate

Learning Parameters / Early Stopping

min-earlystop$(f, \mathcal{D}^{\text{train}}, \theta_0, \ell, \mathcal{D}^{\text{val}}, I_{\text{eval}}, I_{\text{patience}}, \ldots)$:
  $\theta := \theta_0$
  $\theta^* := \theta, i^* := 0, L^* := \ell(\theta; \mathcal{D}^{\text{val}})$
  for $i \in (1, 2, 3, \ldots)$ while $i - i^* \leq I_{\text{patience}}$:
    $\theta :=$ min-fixed-iterations$(f, \mathcal{D}^{\text{train}}, \theta, I_{\text{eval}}, \ldots)$
    $L := \ell(\theta; \mathcal{D}^{\text{val}})$
    if $L < L^*$:
      $\theta^* := \theta, i^* := i, L^* := L$
  return θ^*, i^*, L^*

where additionally

- $\ell : \mathbb{R}^K \times (\mathbb{R}^{M \times O})^* \to \mathbb{R}$ loss function; usually $f(\theta, \mathcal{D}) = \ell(\theta, \mathcal{D}) + \lambda\Omega(\theta)$
- $\mathcal{D}^{\text{val}} \in (\mathbb{R}^{M \times O})^*$ validation dataset
- $I_{\text{eval}} \in \mathbb{N}$ number of iterations between evaluations
- $I_{\text{patience}} \in \mathbb{N}$ maximal number of iterations without improvement of validation losses

Early Stopping as a Regularizer

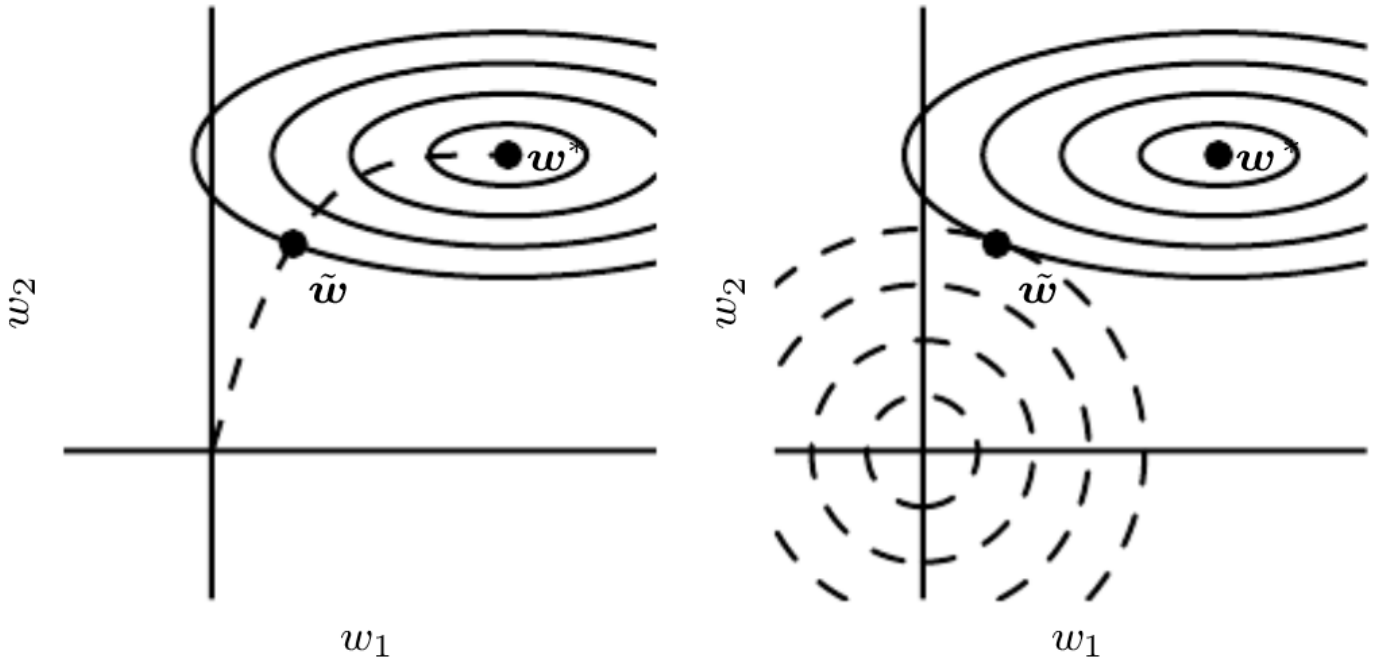


Figure 4: Effect of early stopping (left) on the parameter weights, compared to L2 regularization (right). Source: Goodfellow et al., 2016

Early Stopping as Hyperparameter Optimization

- early stopping also can be seen as hyperparameter optimization:
  - hyperparameter: number i of training iterations
- not relevant in optimization,
  because there models are trained until convergence.
- but relevant in machine learning
  for models that are not perfectly regularized.
- choosing the iteration i^* to stop based on validation loss
  is just a complete search on hyperparameter i
  - overdoing it and training for way to many iterations does not matter
- patience is a mere scalability technique
  - esp. useful for models that are expensive to train such as DNNs

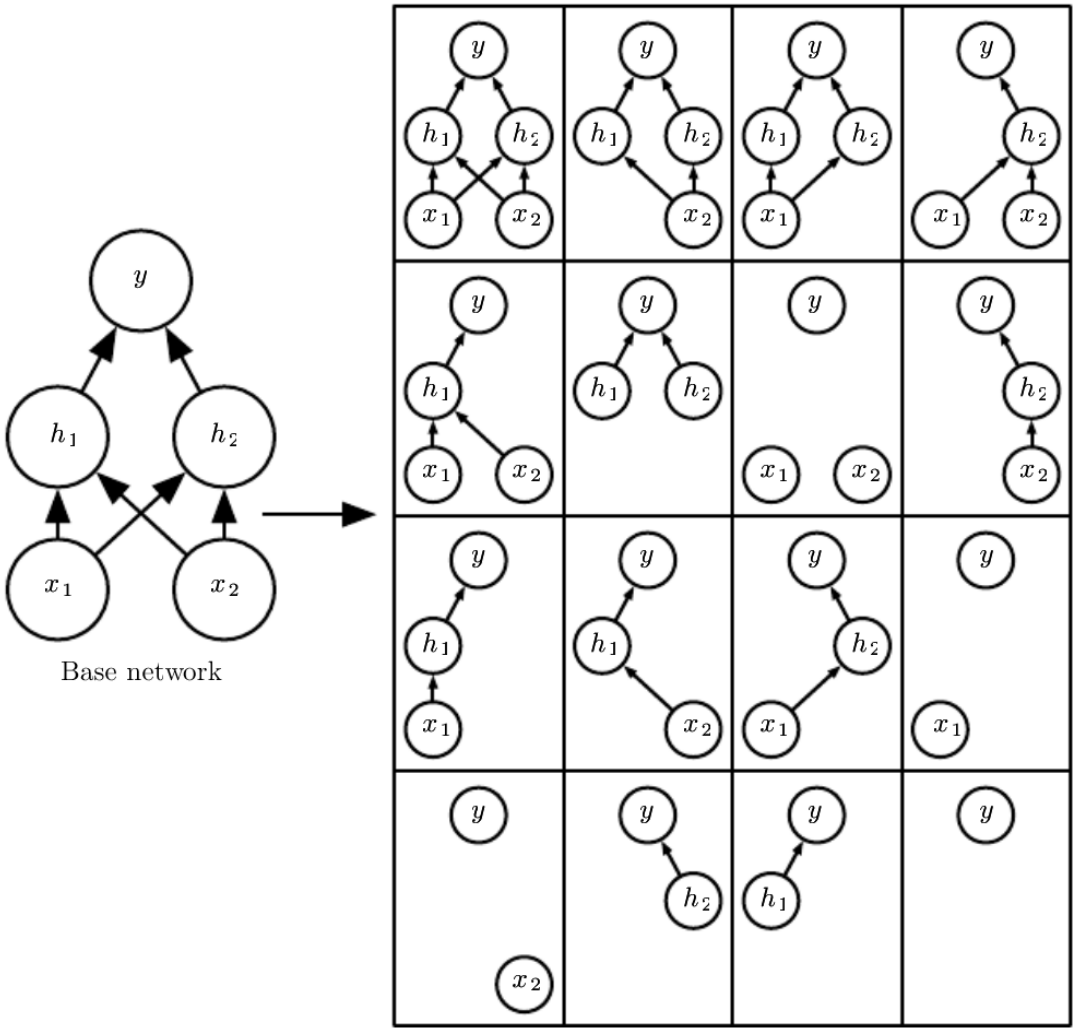
Early Stopping as Hyperparameter Optimization

- Q: Compared to other hyperparameter optimizations as e.g., layer size, is the number of training iterations cheap or expensive to optimize?

Outline

1. Overfitting and Underfitting
2. Parameter Shrinkage
3. Early Stopping
4. Dropout
5. Data Augmentation

Dropout / Random Subnetworks



[source: Goodfellow et al. 2016, p. 258]

Dropout Mechanism

- drop input and hidden nodes of a network randomly
  [Hinton et al., 2012; Srivastava et al., 2014]
- represent dropping by a **binary node mask** d^ℓ:
  multiply the value of dropped nodes by zero

$$z^\ell(z^{\ell-1}) := a(W^\ell z^{\ell-1} + b^\ell)$$

$$\rightsquigarrow \quad z^\ell(z^{\ell-1}) := a(W^\ell(d^{\ell-1} \odot z^{\ell-1}) + b^\ell), \quad d^{\ell-1} \in \{0,1\}^{M_{\ell-1}}$$

- randomly draw dropout masks for every minibatch: $d^\ell \sim \mathrm{ber}(M^\ell, p_\ell)$
  - typically, node inclusion probability $p_{\text{input}} := p_0 := 0.8$ for inputs and $p_{\text{hidden}} := p_\ell := 0.5$ for hidden nodes ($\ell \geq 1$)
  - $p_{\text{input}}, p_{\text{hidden}}$ are hyperparameters

Note: $\odot$ denotes elementwise multiplication of vectors: $x \odot y := (x_n y_n)_{n=1:N}, \quad x, y \in \mathbb{R}^N$.

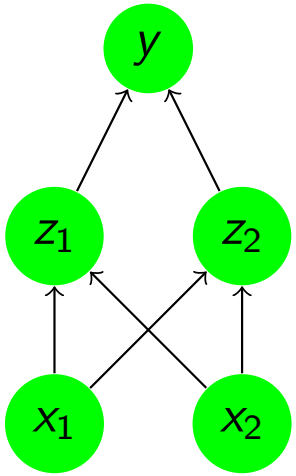
Dropout Mechanism / Backpropagation

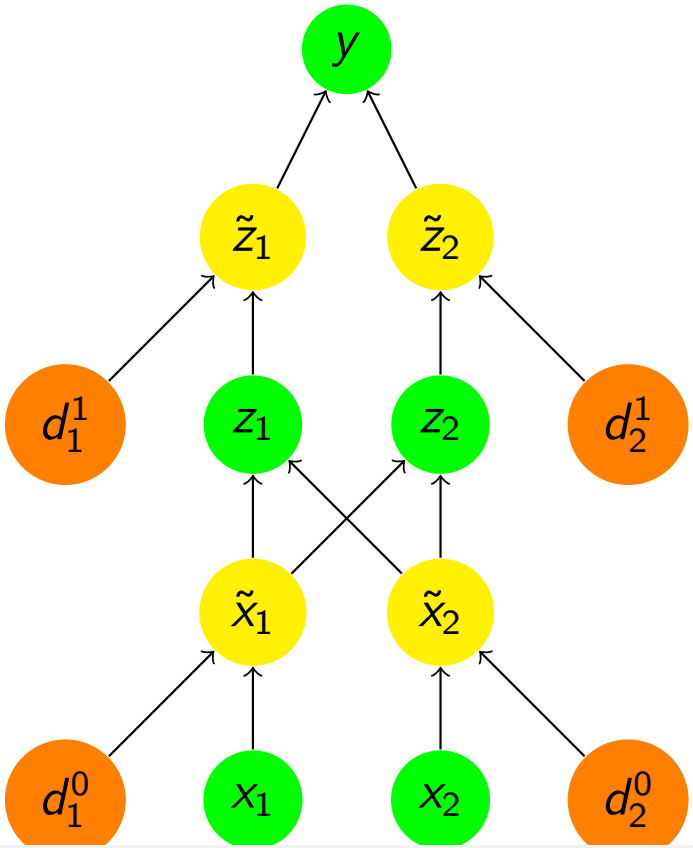
- explicit formula:

$$\frac{\partial z^\ell}{\partial z^{\ell-1}} = \frac{\partial z^\ell}{\partial \tilde{z}^{\ell-1}} \text{diag}(d^{\ell-1}), \quad z^\ell(\tilde{z}^{\ell-1}) := a(W^\ell \tilde{z}^{\ell-1} + b^\ell)$$

$$\tilde{z}^{\ell-1}(z^{\ell-1}) := d^{\ell-1} \odot z^{\ell-1}$$

- or just use the computational graph:





Dropout / Inference

- when used for inference after training,
  average over all possible masks:

$$\tilde{z} = \mathbb{E}_{d \sim \text{ber}(M,p)}(d \odot z) = \ldots$$

- Q: what are the expected latent values under dropout with a node inclusion probability of p?

Dropout / Inference

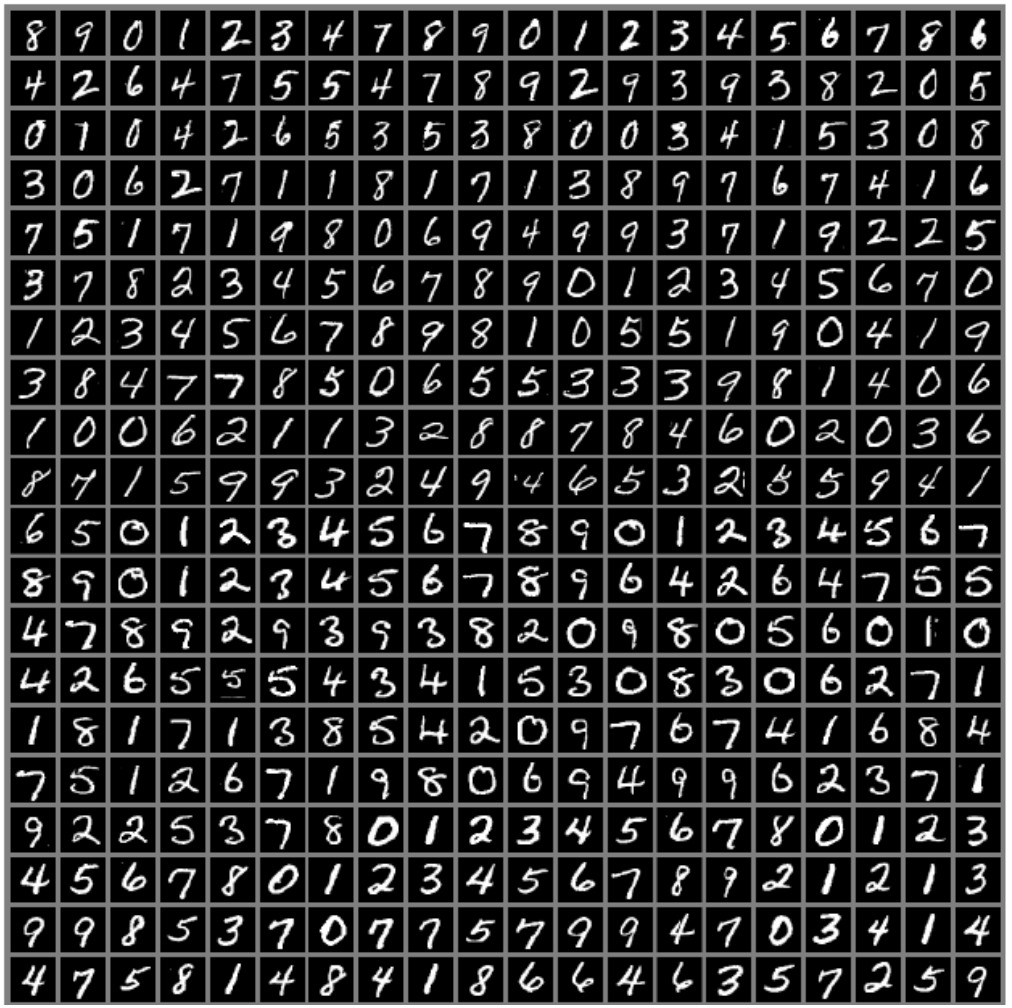
- when used for inference after training,
  average over all possible masks:

$$\tilde{z} = \mathbb{E}_{d \sim \text{ber}(M,p)}(d \odot z) = p\, z$$

- easily implemented in the network without masks:
  - all nodes are used, none is dropped
  - all weights are scaled down by the node inclusion probability of the nodes they are multiplied with:

$$W^{\ell,\text{final}} := p_{\ell-1} W^{\ell}$$

Dropout / Experiment MNIST



[source: Goodfellow et al. 2016, p. 22]

Dropout / Experiment MNIST

- data: MNIST
  - $N = 60$k training plus 10k test instances with
  - $M = 784$ predictors (28×28 images of handwritte digits) and
  - $O = 10$ targets (binary indicators for digits $\{0, 1, 2, \ldots, 9\}$)
- models: feedforward neural network
  - ignore spatial arrangement of the predictors
  - 2 layers of size 800:
    $(M + 1)800 + (800 + 1)800 + (800 + 1)O = 1.3$M parameters

    $\vdots$
  - 3 layers of size 2048: 10M parameters
  - 2 layers of size 8192: 74M parameters

Dropout / Results MNIST

Method	Unit Type	Architecture	Error %
Standard Neural Net (Simard et al., 2003)	Logistic	2 layers, 800 units	1.60
SVM Gaussian kernel	NA	NA	1.40
Dropout NN	Logistic	3 layers, 1024 units	1.35
Dropout NN	ReLU	3 layers, 1024 units	1.25
Dropout NN + max-norm constraint	ReLU	3 layers, 1024 units	1.06
Dropout NN + max-norm constraint	ReLU	3 layers, 2048 units	1.04
Dropout NN + max-norm constraint	ReLU	2 layers, 4096 units	1.01
Dropout NN + max-norm constraint	ReLU	2 layers, 8192 units	0.95
Dropout NN + max-norm constraint (Goodfellow et al., 2013)	Maxout	2 layers, (5×240) units	0.94
DBN + finetuning (Hinton and Salakhutdinov, 2006)	Logistic	500-500-2000	1.18
DBM + finetuning (Salakhutdinov and Hinton, 2009)	Logistic	500-500-2000	0.96
DBN + dropout finetuning	Logistic	500-500-2000	0.92
DBM + dropout finetuning	Logistic	500-500-2000	**0.79**

Table 2: Comparison of different models on MNIST.

Note: DBN = deep belief network, DBM = deep boltzmann machine, max-norm constraint see slide 11.

[source: Srivastava et al. 2014, p. 1936]

Dropout / Results MNIST

Method	**Test Classification error %**
L2	1.62
L2 + L1 applied towards the end of training	1.60
L2 + KL-sparsity	1.55
Max-norm	1.35
Dropout + L2	1.25
Dropout + Max-norm	**1.05**

Table 9: Comparison of different regularization methods on MNIST.

[source: Srivastava et al. 2014, p. 1943]

Why does Dropout Work?

- co-adaptation: nodes extract features that are useful only with the help of features extracted by other neurons
  - dropout reduces co-adaptation as neurons cannot rely on all other neurons being present, but just a random subset.
- ensembles:
  - ensembles = a collection of models used together, e.g., thousand decision trees, using their average prediction as prediction of the ensemble.
  - selecting a random subset of predictors is a well-known ensembling technique called bagging (see ML2 lecture).
  - using random subnetworks introduces heterogeneity by using different network wirings.
  - but finally
    - parameters of all these ensemble components are averaged,
    - a single model including all nodes is used.

Outline

1. Overfitting and Underfitting

2. Parameter Shrinkage

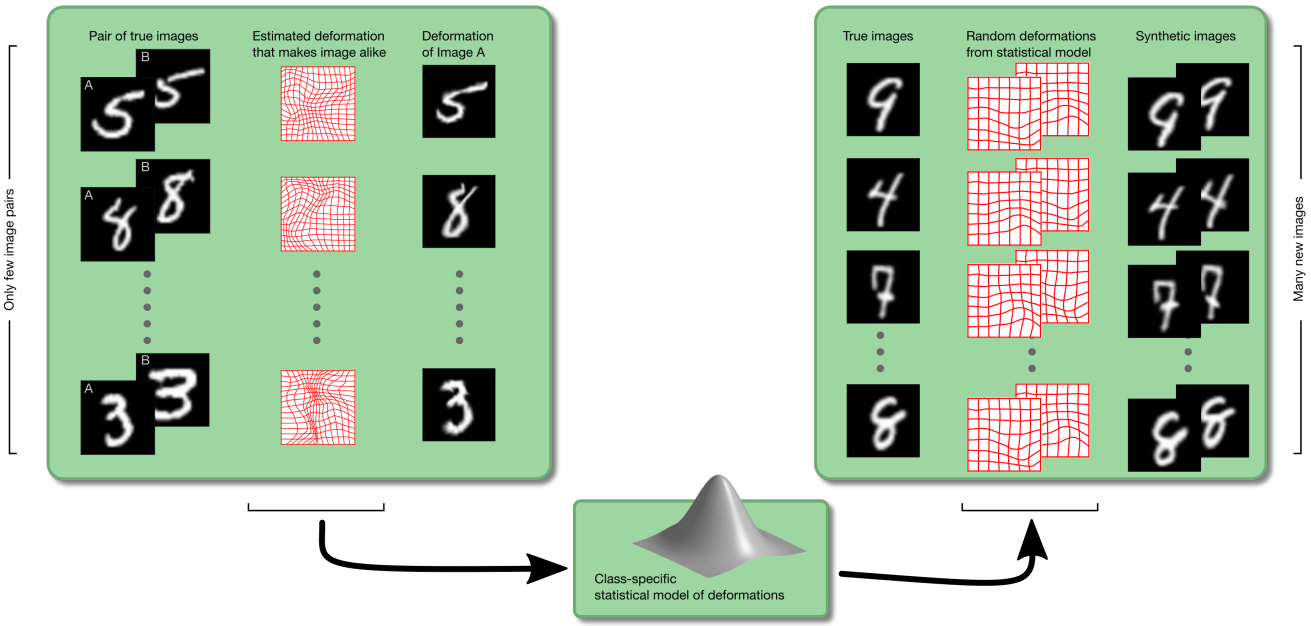
3. Early Stopping

4. Dropout

5. Data Augmentation

Data Augmentation (Noise to Input)

- Train a model with more data to improve generalization
- Create additional synthetic, "fake" data
  by perturbing existing training set instances
- e.g., for image classification:
  - translation, rotation, scaling of images; or deformation strategies:



[source: compute.dtu.dk]

Data Augmentation

- learn from two training sets:
  - the original training set: $\mathcal{D}^{\text{train}}$
  - the set of augmented samples: $\mathcal{D}^{\text{train}+} := \text{augment}(\mathcal{D}^{\text{train}})$
- discount the influence of the augmented samples by a case weight:

$$f(\theta; \mathcal{D}^{\text{train}}, \mathcal{D}^{\text{train}+}) := \ell(\theta; \mathcal{D}^{\text{train}}) + \alpha\ell(\theta; \mathcal{D}^{\text{train}+}) + \lambda\Omega(\theta), \quad \alpha \in \mathbb{R}_0^+$$

  - augmented sample weight α is a hyperparameter

Robustness Against Parameter Noise

- noise to parameters reduces overfitting
- used primarily with recurrent neural networks
- consider a regression problem:

$$f(\theta) = \mathbb{E}_{(x,y)\sim p}\left((\hat{y}(x;\theta) - y)^2\right)$$

- adding a perturbation $\Delta\theta \sim \mathcal{N}(0, \eta I)$ to the parameters yields a perturbed prediction $\hat{y}(x, \theta + \Delta\theta)$, such that:

$$f(\theta) = \mathbb{E}_{\Delta\theta\sim\mathcal{N}(0,\eta I)}\,\mathbb{E}_{(x,y)\sim p}\left((\hat{y}(x;\theta + \Delta\theta) - y)^2\right)$$

- the optimization of this objective for small η is equivalent to adding a penalty $\eta\,\mathbb{E}_{(x,y)\sim p}\left(||\nabla_\theta \hat{y}(x;\theta)||^2\right)$

Summary (1/2)

- **Overfitting** the training data too closely leads to bad generalization on new data and must be avoided (**regularization**).
- Models can be regularized by adding a penalty term for large parameter values to the objective function (**parameter shrinkage**)
  - simply the sum of squares of all parameters (**L_2 regularization**) or the sum of their absolute values (**L_1 regularization**)
- Also **early stopping**, i.e., stop training once the error on a validation sample starts to deteriorate, is a simple regularization method.
- Creating more synthetic data, i.e., by adding transformations or noise to predictors, is another simple regularization method (**data augmentation**).

Summary (2/2)

- **Dropout** is a regularization technique specific for neural networks using random subnetworks:
  - for each minibatch, a random set of nodes is dropped (forced to zero)
  - droppings are represented by **binary node masks**
  - predicton and backprogation can easily be incorporate the node masks

Further Readings

- Goodfellow et al. 2016, ch. 7
- Zhang et al. 2020, ch. 4.4–7
- lecture Machine Learning, chapter A.3

Acknowledgement: An earlier version of the slides for this lecture have been written by my former postdoc Dr Josif Grabocka.

References

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The Mit Press, Cambridge, Massachusetts, November 2016. ISBN 978-0-262-03561-3.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander Smola. *Dive into Deep Learning*. https://d2l.ai/, 2020.