

Network Regularization

Weight initialization, dropout, batch normalization

Machine Learning and Data Mining, 2020

Artem Maevskiy

National Research University Higher School of Economics



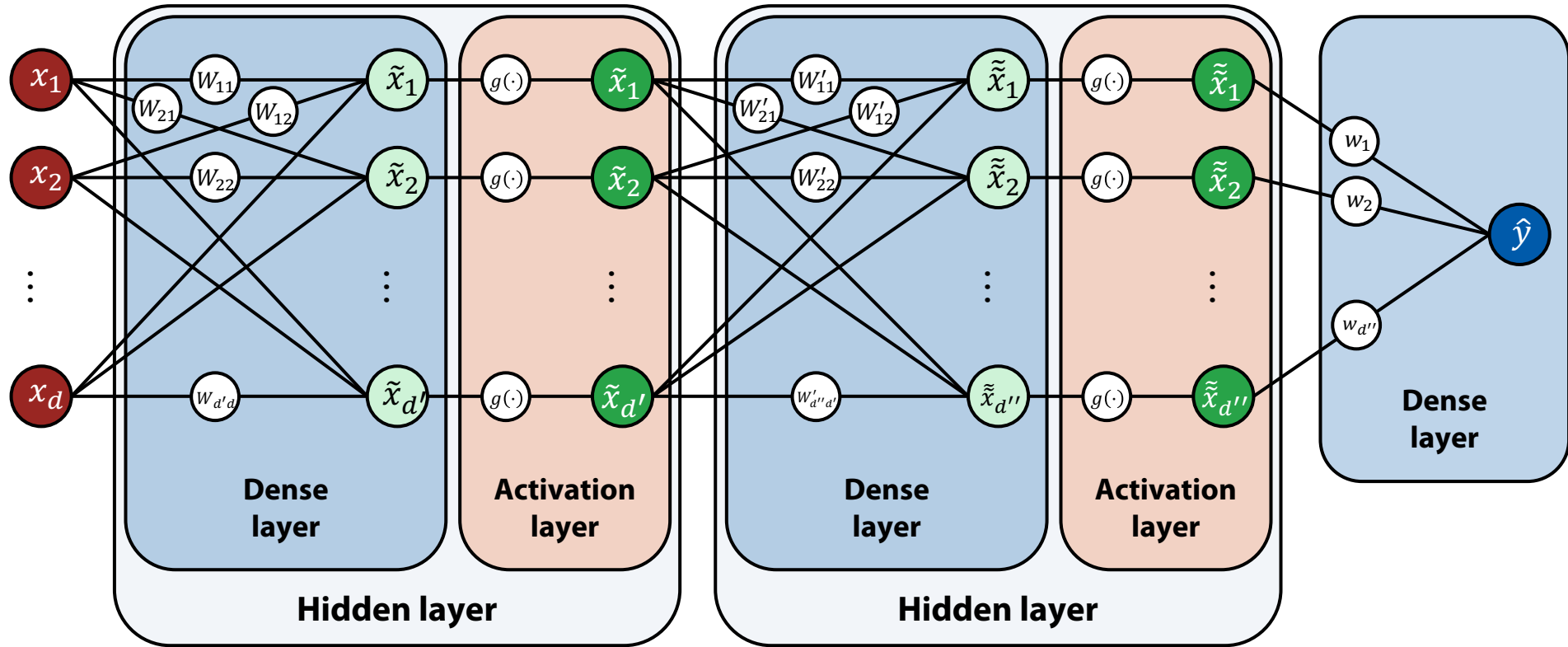
LAMBA • HSE

November 13, 2020

Why care about weight initialization?

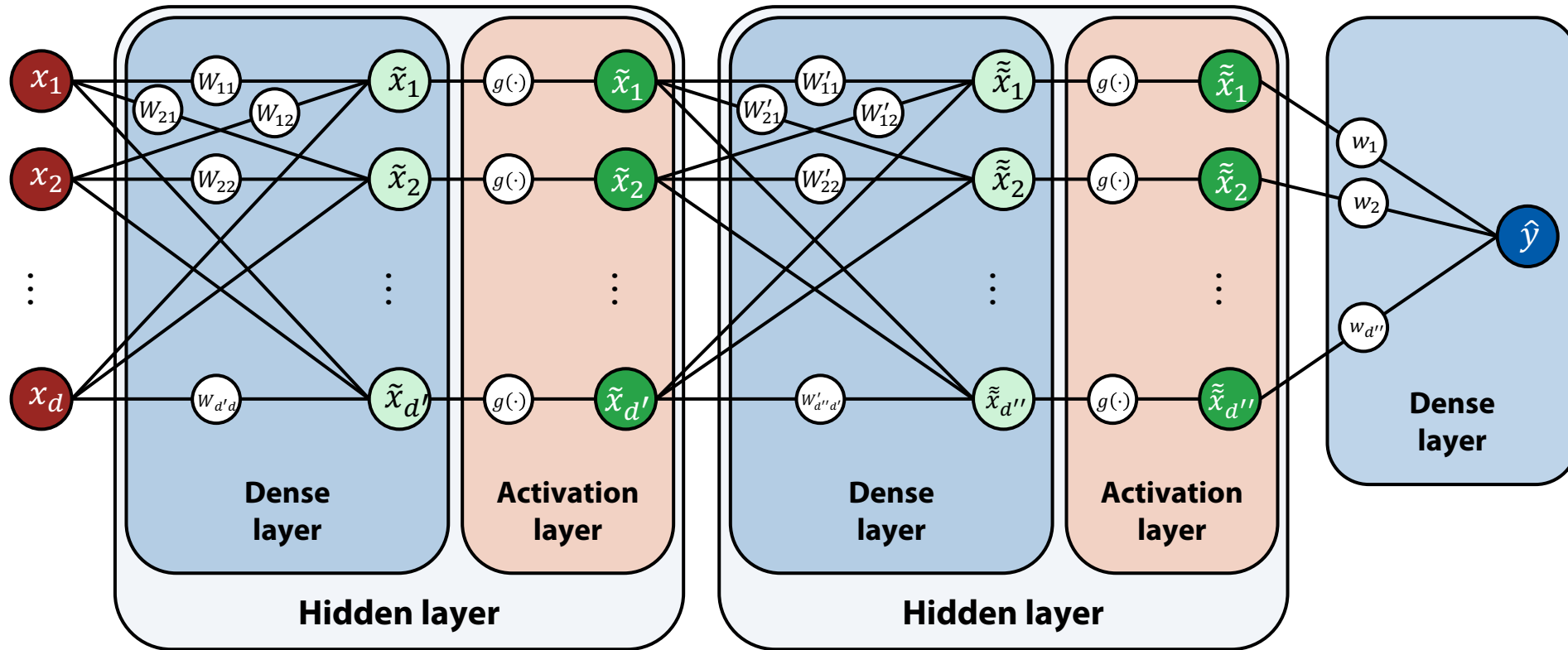


Initialization with a constant (?)



- What happens if we initialize all weights with the same value?

Initialization with a constant (?)



- ▶ What happens if we initialize all weights with the same value?
- ▶ Within each layer, the gradients for each of the weights will be the same as well \Rightarrow **updates will be the same** \Rightarrow network degrades!

Initialization with a constant (?)

- ▶ Ok, so constant initialization is a bad idea
- ▶ So, any random initialization should be fine, right?

Some intuition

- ▶ For simplicity, let's omit the activation functions for now
- ▶ Then, the output of a neural network composed of dense layers only is:

$$\hat{y} = W_{out} \cdot \dots \cdot W_{h2} \cdot W_{h1}x$$

Some intuition

- ▶ For simplicity, let's omit the activation functions for now
- ▶ Then, the output of a neural network composed of dense layers only is:

$$\hat{y} = W_{out} \cdot \dots \cdot W_{h2} \cdot W_{h1}x$$

- ▶ Note that gradient wrt to any of the weight matrices W_{hk} is proportional to the **product** of all other matrices

Some intuition

- ▶ For simplicity, let's omit the activation functions for now
- ▶ Then, the output of a neural network composed of dense layers only is:

$$\hat{y} = W_{out} \cdot \dots \cdot W_{h2} \cdot W_{h1}x$$

- ▶ Note that gradient wrt to any of the weight matrices W_{hk} is proportional to the **product** of all other matrices
- ▶ E.g. for 1×1 matrices, if all are of scale $S \in \mathbb{R}$, the gradient g is proportional to:

$$g \sim S^{m-1}$$

where m is the **depth** of the network

Some intuition

- ▶ For simplicity, let's omit the activation functions for now
- ▶ Then, the output of a neural network composed of dense layers only is:

$$\hat{y} = W_{out} \cdot \dots \cdot W_{h2} \cdot W_{h1}x$$

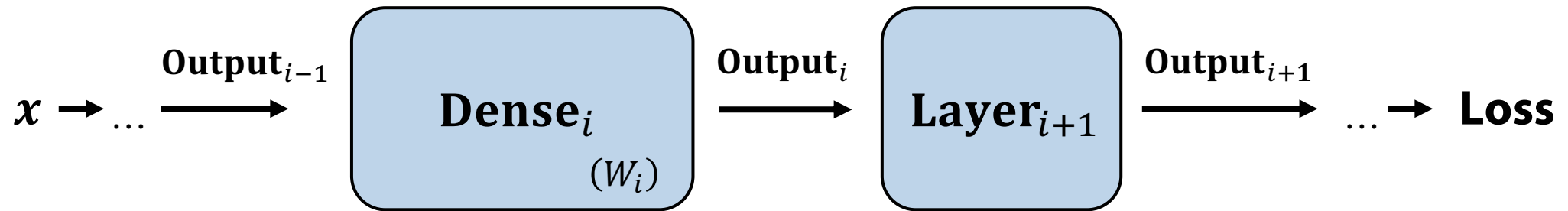
- ▶ Note that gradient wrt to any of the weight matrices W_{hk} is proportional to the **product** of all other matrices
- ▶ E.g. for 1×1 matrices, if all are of scale $S \in \mathbb{R}$, the gradient g is proportional to:

$$g \sim S^{m-1}$$

where m is the **depth** of the network

- ▶ For S too large, the gradients will **explode**; for S too small, they will **vanish**

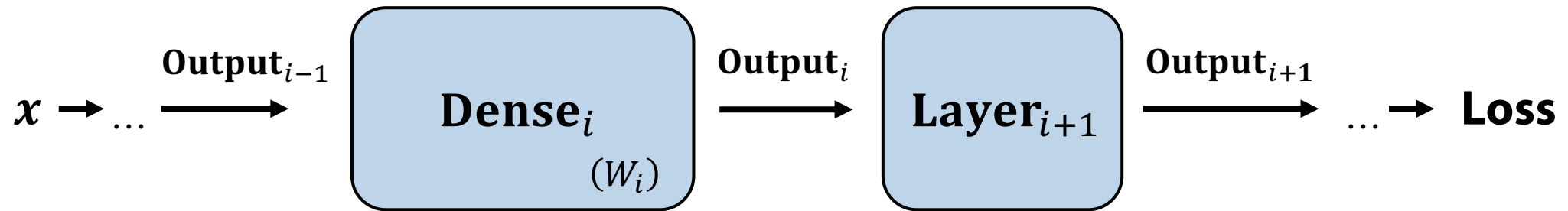
Some intuition



- More generally:

$$\frac{\partial \text{Loss}}{\partial W_i} = \frac{\partial \text{Loss}}{\partial \text{Output}_i} \cdot \frac{\partial \text{Dense}_i}{\partial W_i} = \frac{\partial \text{Loss}}{\partial \text{Output}_{i+1}} \cdot \frac{\partial \text{Layer}_{i+1}}{\partial \text{Output}_i} \cdot \text{Output}_{i-1}$$

Some intuition

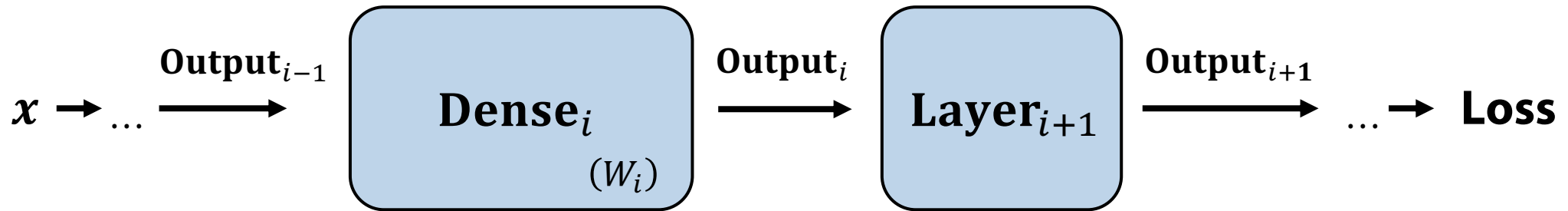


- More generally:

$$\frac{\partial \text{Loss}}{\partial W_i} = \frac{\partial \text{Loss}}{\partial \text{Output}_i} \cdot \frac{\partial \text{Dense}_i}{\partial W_i} = \underbrace{\frac{\partial \text{Loss}}{\partial \text{Output}_{i+1}} \cdot \frac{\partial \text{Layer}_{i+1}}{\partial \text{Output}_i}}_{\text{This will accumulate the product of the gradients for the subsequent layers}} \cdot \text{Output}_{i-1}$$

This will accumulate the product of the gradients for the subsequent layers

Some intuition



- More generally:

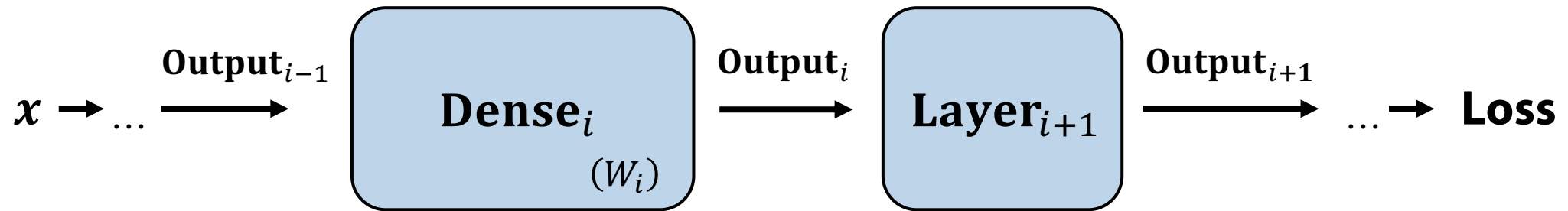
$$\frac{\partial \text{Loss}}{\partial W_i} = \frac{\partial \text{Loss}}{\partial \text{Output}_i} \cdot \frac{\partial \text{Dense}_i}{\partial W_i} = \frac{\partial \text{Loss}}{\partial \text{Output}_{i+1}} \cdot \frac{\partial \text{Layer}_{i+1}}{\partial \text{Output}_i} \cdot \text{Output}_{i-1}$$

This will accumulate the product of the gradients for the subsequent layers

- Idea: for stable learning we would like to “keep” the scale of the gradients at each step:

$$\text{Var} \left(\frac{\partial \text{Layer}_{i+1}}{\partial \text{Output}_i} \cdot \frac{\partial \text{Layer}_i}{\partial \text{Output}_{i-1}} \right) \approx \text{Var} \left(\frac{\partial \text{Layer}_{i+1}}{\partial \text{Output}_i} \right)$$

Some intuition



- ▶ Similarly, we would also like to not scale the outputs at each step of the forward pass:

$$\text{Var}\left(\text{Layer}_{i+1}\left(\text{Layer}_i\left(\text{Output}_{i-1}\right)\right)\right) \approx \text{Var}\left(\text{Layer}_i\left(\text{Output}_{i-1}\right)\right)$$

Random initialization

$$\text{Var} \left(\frac{\partial \mathbf{Layer}_{i+1}}{\partial \mathbf{Output}_i} \cdot \frac{\partial \mathbf{Layer}_i}{\partial \mathbf{Output}_{i-1}} \right) \approx \text{Var} \left(\frac{\partial \mathbf{Layer}_{i+1}}{\partial \mathbf{Output}_i} \right)$$

$$\text{Var} \left(\mathbf{Layer}_{i+1}(\mathbf{Layer}_i(\mathbf{Output}_{i-1})) \right) \approx \text{Var}(\mathbf{Layer}_i(\mathbf{Output}_{i-1}))$$

Random initialization

$$\text{Var}\left(\frac{\partial \mathbf{Layer}_{i+1}}{\partial \mathbf{Output}_i} \cdot \frac{\partial \mathbf{Layer}_i}{\partial \mathbf{Output}_{i-1}}\right) \approx \text{Var}\left(\frac{\partial \mathbf{Layer}_{i+1}}{\partial \mathbf{Output}_i}\right)$$

$$\text{Var}\left(\mathbf{Layer}_{i+1}(\mathbf{Layer}_i(\mathbf{Output}_{i-1}))\right) \approx \text{Var}(\mathbf{Layer}_i(\mathbf{Output}_{i-1}))$$

- ▶ Generally, these two requirements may contradict each other
- ▶ E.g. for ReLU activation they result in initialization requirements, respectively:

$$\text{Var}(W_{ij}) = \frac{2}{(\# \text{ outgoing connections})}$$

$$\text{Var}(W_{ij}) = \frac{2}{(\# \text{ incoming connections})}$$

Random initialization

$$\text{Var}\left(\frac{\partial \mathbf{Layer}_{i+1}}{\partial \mathbf{Output}_i} \cdot \frac{\partial \mathbf{Layer}_i}{\partial \mathbf{Output}_{i-1}}\right) \approx \text{Var}\left(\frac{\partial \mathbf{Layer}_{i+1}}{\partial \mathbf{Output}_i}\right)$$

$$\text{Var}\left(\mathbf{Layer}_{i+1}(\mathbf{Layer}_i(\mathbf{Output}_{i-1}))\right) \approx \text{Var}(\mathbf{Layer}_i(\mathbf{Output}_{i-1}))$$

- ▶ Generally, these two requirements may contradict each other
- ▶ E.g. for ReLU activation they result in initialization requirements, respectively:

$$\text{Var}(W_{ij}) = \frac{2}{(\# \text{ outgoing connections})}$$

$$\text{Var}(W_{ij}) = \frac{2}{(\# \text{ incoming connections})}$$

- ▶ Typically you can just choose one of them, or alternatively average them out:

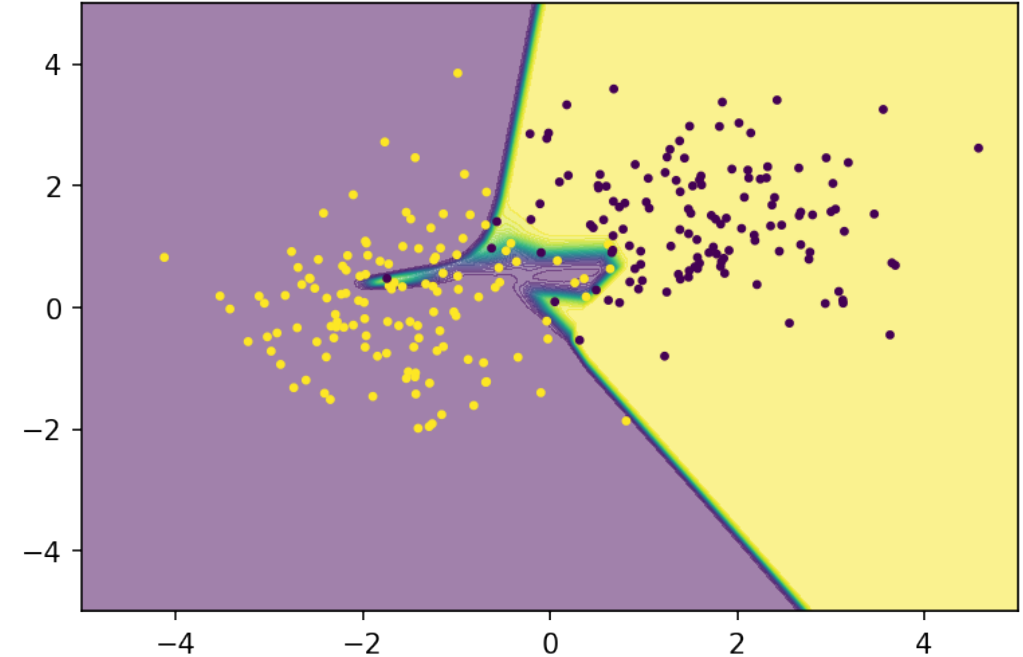
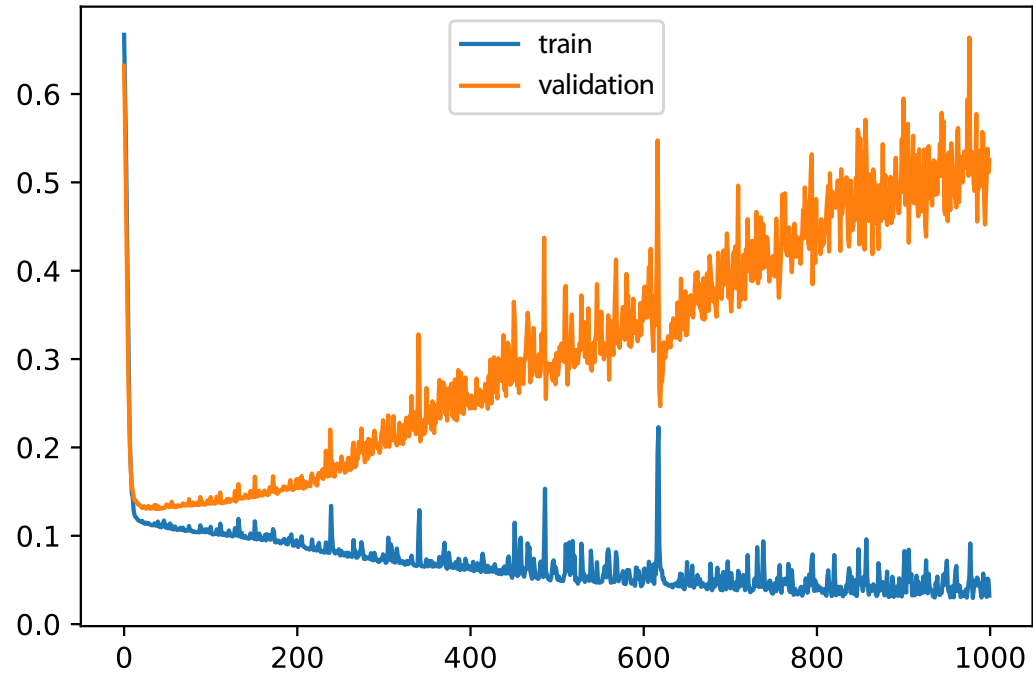
$$\text{Var}(W_{ij}) = \frac{4}{(\# \text{ outgoing connections}) + (\# \text{ incoming connections})}$$

Overfitting with neural networks



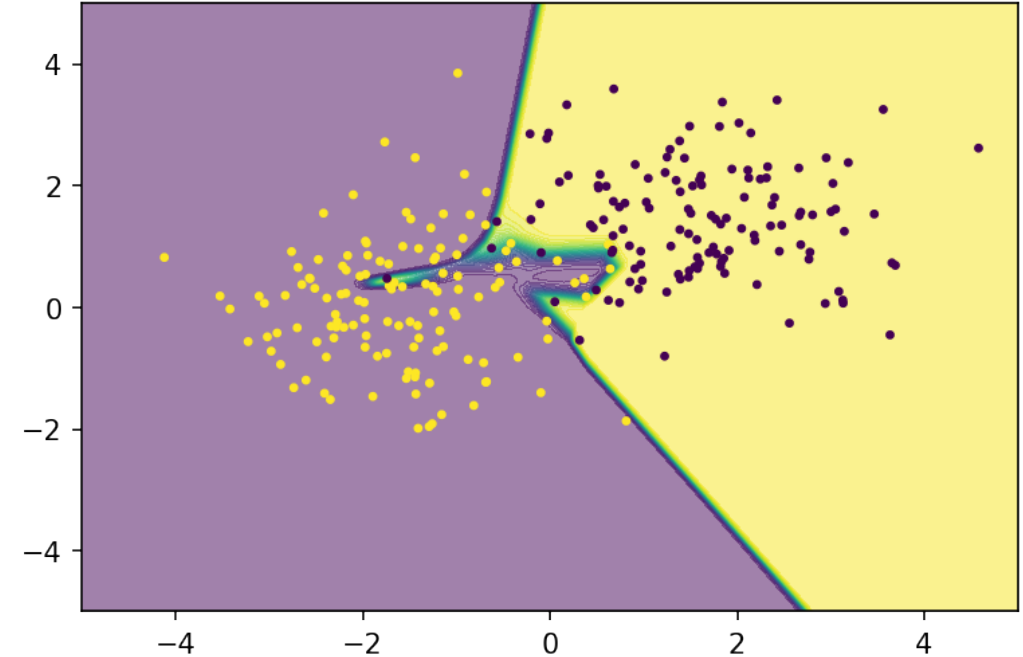
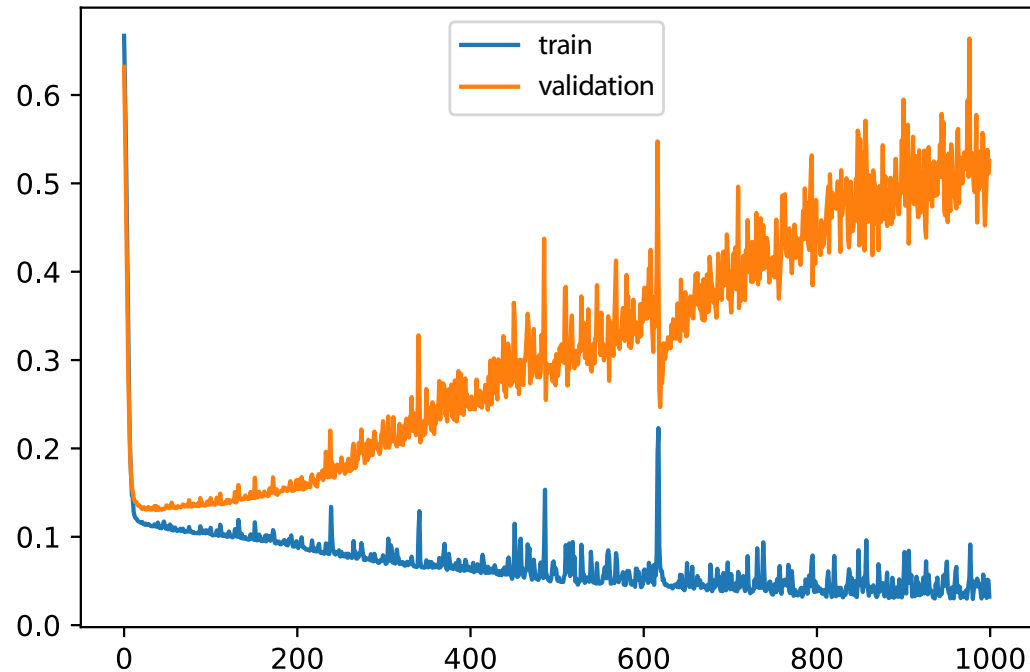
The problem of overfitting

- Being highly complex models, neural networks are prone to overfitting



The problem of overfitting

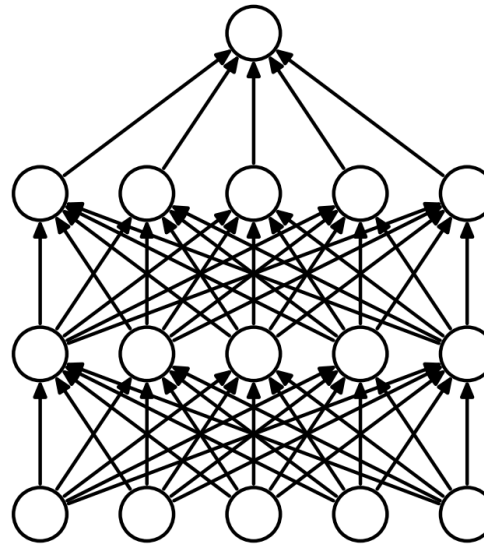
- ▶ Being highly complex models, neural networks are prone to overfitting



- ▶ Regularization techniques like L1/L2 regularization are also available for neural networks
- ▶ We also discussed **early stopping** (i.e. stop the training before validation error grows)

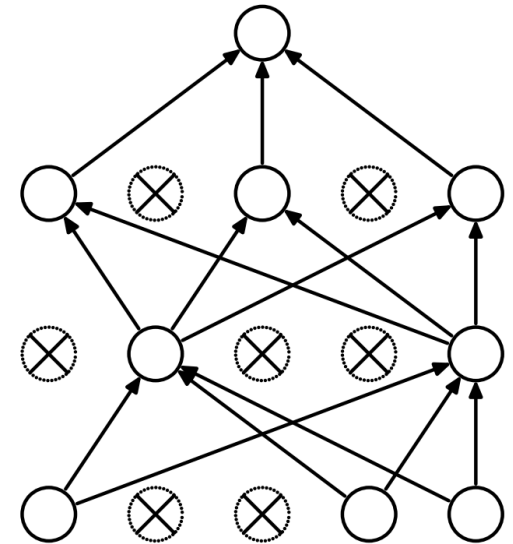
Dropout

- ▶ At train time – sets neuron activations to 0 with a given probability p



(a) Standard Neural Net

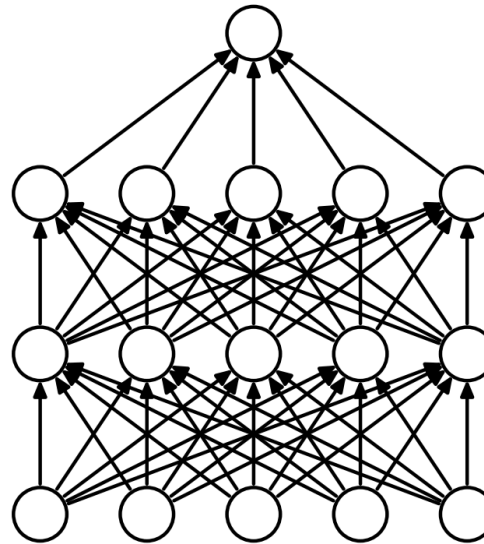
Image from:
<http://jmlr.org/papers/v15/srivastava14a.html>



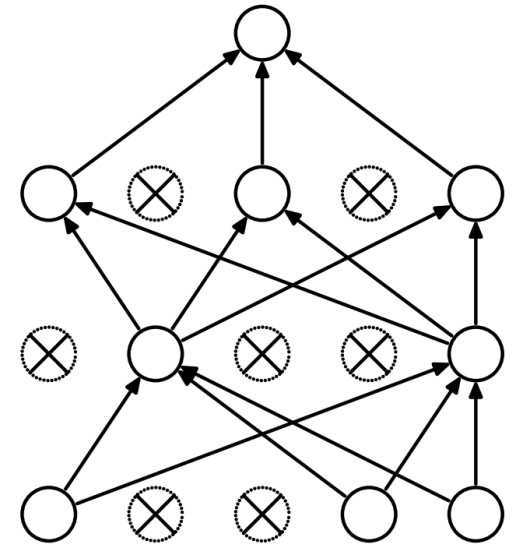
(b) After applying dropout.

Dropout

- ▶ At train time – sets neuron activations to 0 with a given probability p
- ▶ At test time – multiplies the activation by p
 - i.e. sets it to the **expected value**



(a) Standard Neural Net

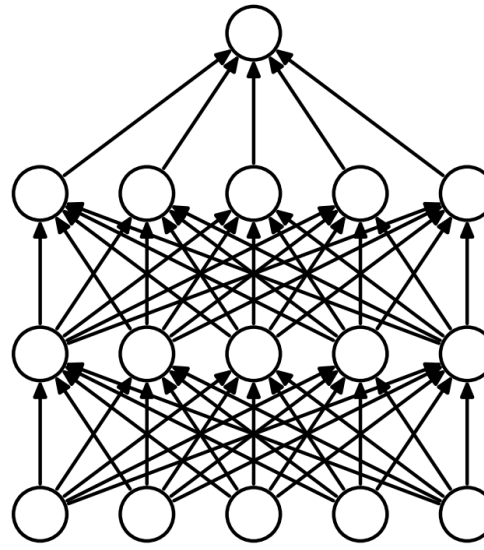


(b) After applying dropout.

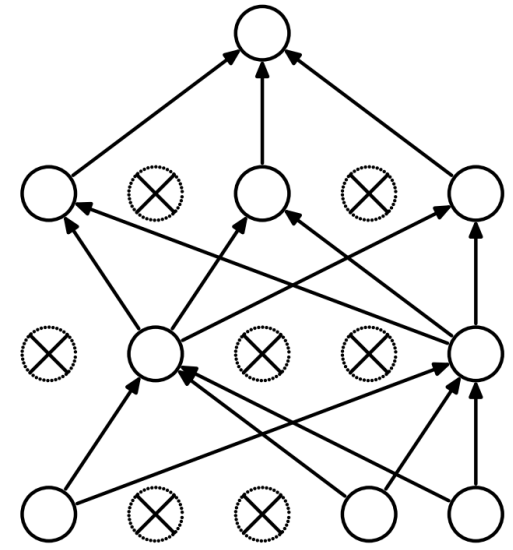
Image from:
<http://jmlr.org/papers/v15/srivastava14a.html>

Dropout

- ▶ At train time – sets neuron activations to 0 with a given probability p
- ▶ At test time – multiplies the activation by p
 - i.e. sets it to the **expected value**
- ▶ Makes neuron learn to work with a **randomly chosen sample** of other neurons



(a) Standard Neural Net

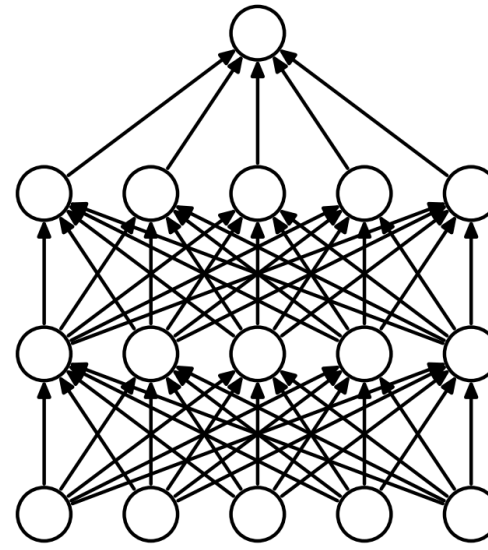


(b) After applying dropout.

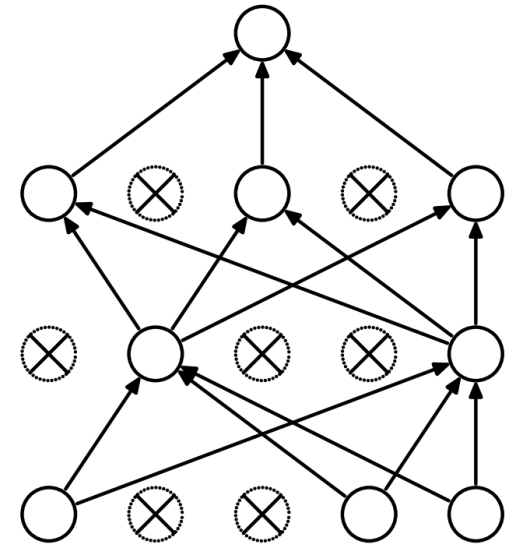
Image from:
<http://jmlr.org/papers/v15/srivastava14a.html>

Dropout

- ▶ At train time – sets neuron activations to 0 with a given probability p
- ▶ At test time – multiplies the activation by p
 - i.e. sets it to the **expected value**
- ▶ Makes neuron learn to work with a **randomly chosen sample** of other neurons
- ▶ Drives it towards **creating useful features** rather than relying on other neurons to correct its mistakes



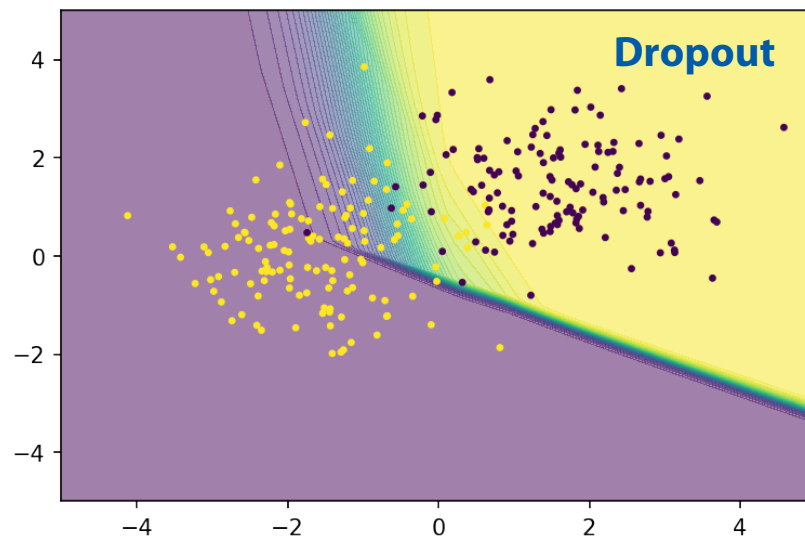
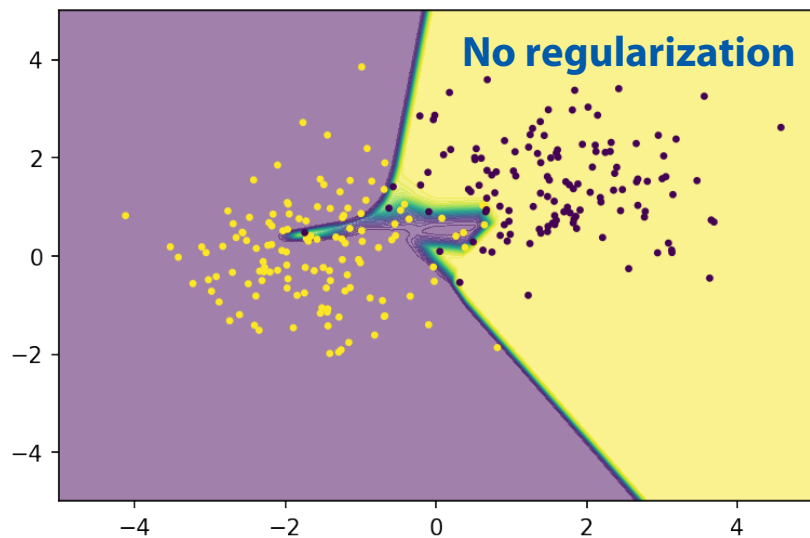
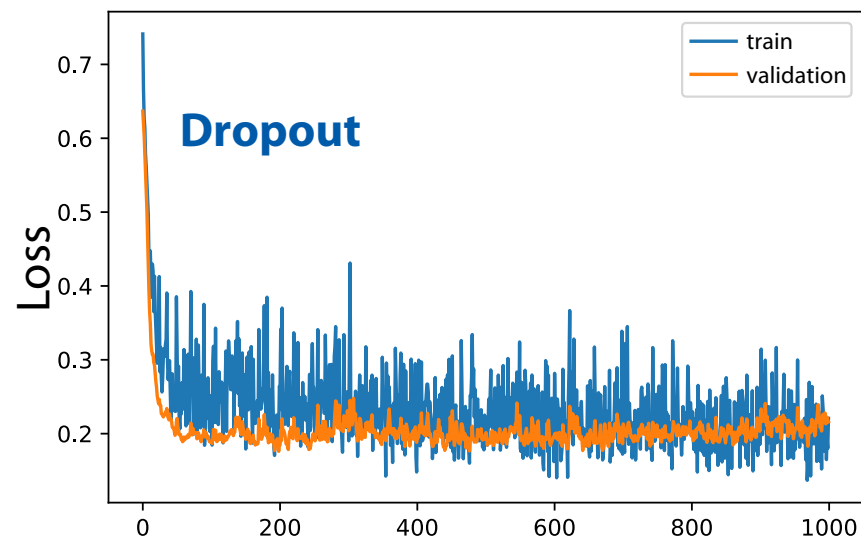
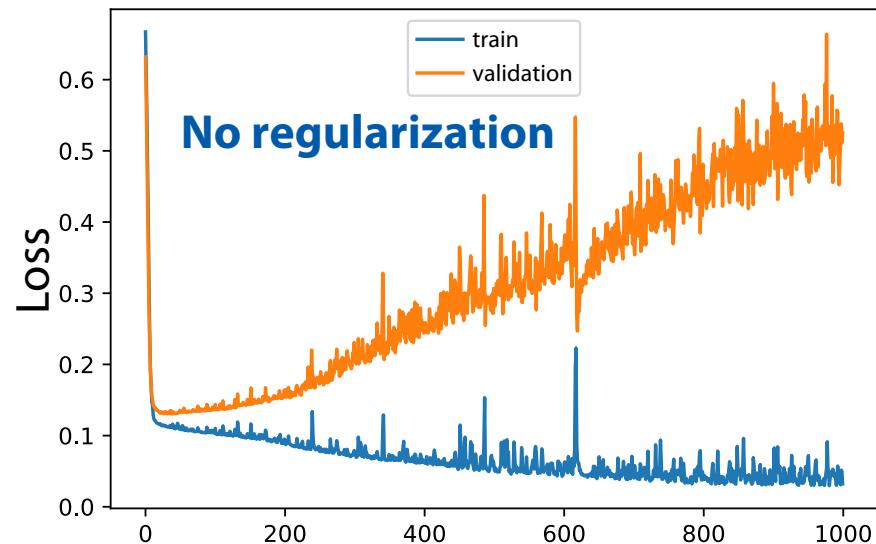
(a) Standard Neural Net



(b) After applying dropout.

Image from:
<http://jmlr.org/papers/v15/srivastava14a.html>

Example from before



In this example, dropout results in a much better (though still not perfect) fit with lower test error

Normalization layers



Batch normalization

- ▶ This technique was originally proposed to mitigate the “internal covariate shift”

internal covariate shift

the updates in one layer change
the input distributions of the
subsequent layers

Batch normalization

- ▶ This technique was originally proposed to mitigate the “internal covariate shift”
- ▶ Works as follows (layer inputs x_i , outputs y_i):
 - calculate sample **mean** and **variance** of the input on a single batch B

internal covariate shift

the updates in one layer change
the input distributions of the
subsequent layers

$$\mu_B = \frac{1}{|B|} \sum_{i \in B} x_i \qquad \sigma_B^2 = \frac{1}{|B|} \sum_{i \in B} (x_i - \mu_B)^2$$

Batch normalization

- ▶ This technique was originally proposed to mitigate the “internal covariate shift”
- ▶ Works as follows (layer inputs x_i , outputs y_i):
 - calculate sample **mean** and **variance** of the input on a single batch B

internal covariate shift

the updates in one layer change
the input distributions of the
subsequent layers

$$\mu_B = \frac{1}{|B|} \sum_{i \in B} x_i \quad \sigma_B^2 = \frac{1}{|B|} \sum_{i \in B} (x_i - \mu_B)^2$$

- **normalize** the input, then **scale and shift** (with the trainable parameters γ, β):

$$y_i = \gamma \cdot \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

Batch normalization

- ▶ Turned out to be **extremely powerful** in many cases
 - Faster and more stable convergence

internal covariate shift

the updates in one layer change
the input distributions of the
subsequent layers

Batch normalization

- ▶ Turned out to be **extremely powerful** in many cases
 - Faster and more stable convergence
- ▶ Later was proved to **not** reduce the internal covariate shift

internal covariate shift

the updates in one layer change
the input distributions of the
subsequent layers

Batch normalization

- ▶ Turned out to be **extremely powerful** in many cases
 - Faster and more stable convergence
- ▶ Later was proved to **not** reduce the internal covariate shift

internal covariate shift

the updates in one layer change
the input distributions of the
subsequent layers

- ▶ Effectively **removes** the 'shift' and 'scale' degrees of freedom from the previous layer

$$y_i = \gamma \cdot \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

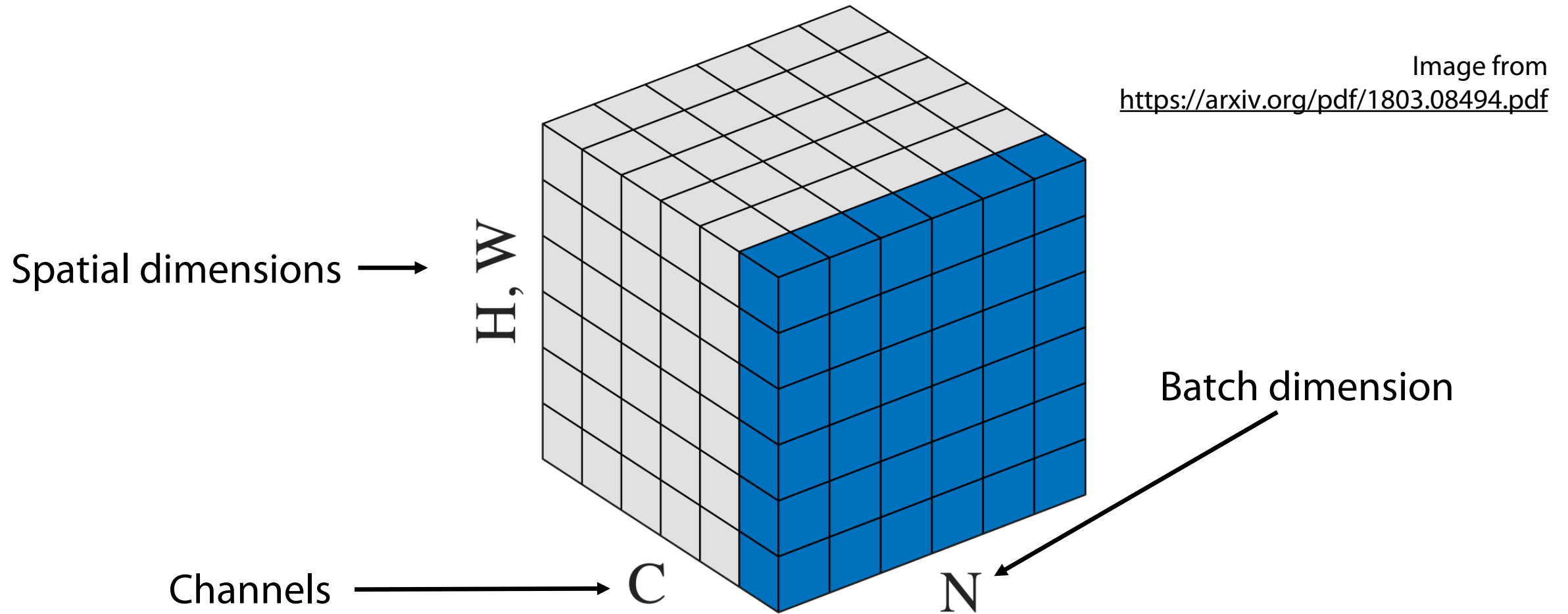
Batch normalization

- ▶ Which dimension to normalize over? Typically like this:
 - Batch of 1D vectors [Batch_dim x Features_dim]
 - separately for each component in Features_dim , i.e. over Batch_dim

Batch normalization

- ▶ Which dimension to normalize over? Typically like this:
 - Batch of 1D vectors [Batch_dim x Features_dim]
 - separately for each component in Features_dim , i.e. over Batch_dim
 - Batch of ND objects [Batch_dim x Spacial_dim1 x ... x Channel_dim]
 - separately for each component in Channel_dim , i.e. over Batch_dim x Spacial_dim1 x ...

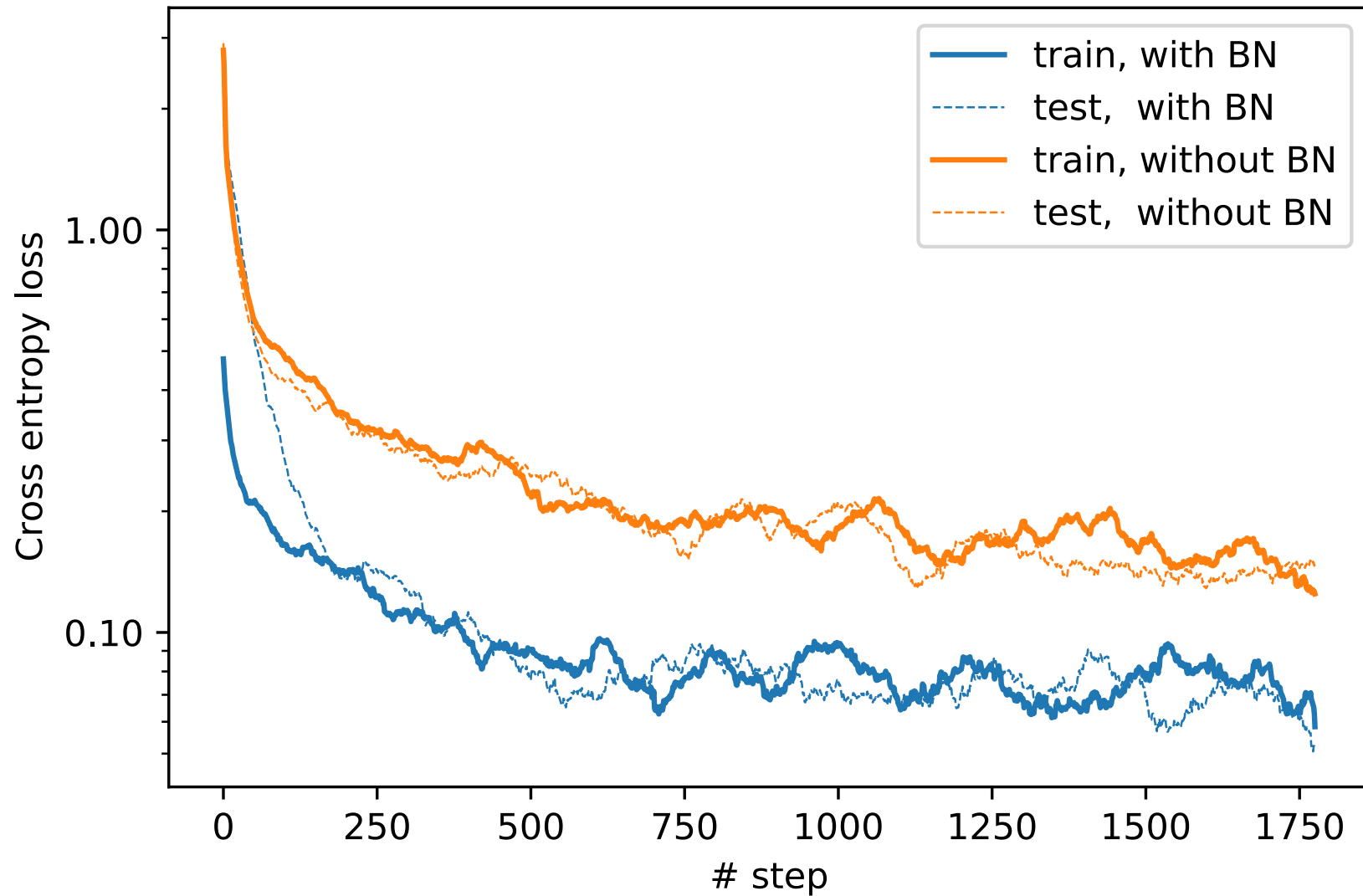
Batch normalization



Batch normalization at inference time

- ▶ Calculating batch statistics at test time may be problematic
 - e.g. when there's a single object to predict
- ▶ Instead: calculate running mean and variance during training, apply at test time

Example: CNN on MNIST



(shown: moving average loss)

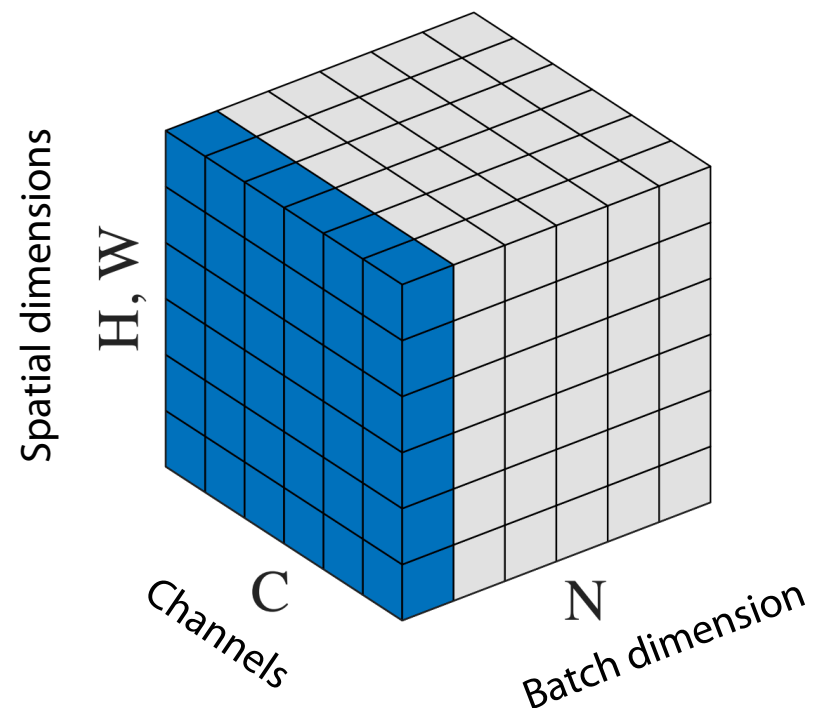
Layer Normalization

- ▶ Batch normalization imposes limits on the batch size
 - if too small, the variance of the sample statistics will be too high
- ▶ Problematic to use in recurrent networks

Layer Normalization

- ▶ Batch normalization imposes limits on the batch size
 - if too small, the variance of the sample statistics will be too high
- ▶ Problematic to use in recurrent networks
- ▶ Alternative: **Layer Normalization**
 - the math is same, except statistics is calculated over channels rather than batch elements

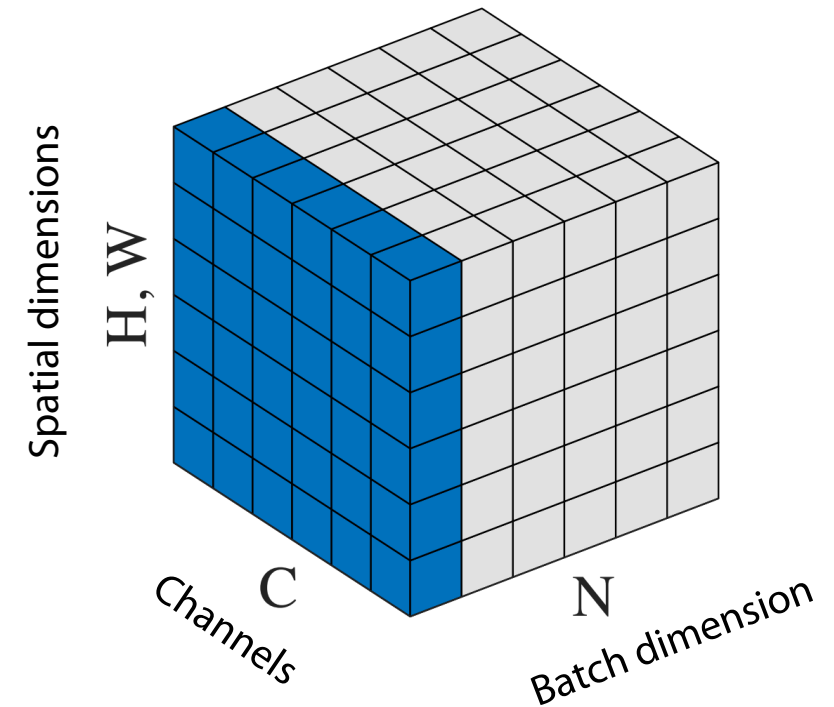
Image from
<https://arxiv.org/pdf/1803.08494.pdf>



Layer Normalization

- ▶ Batch normalization imposes limits on the batch size
 - if too small, the variance of the sample statistics will be too high
- ▶ Problematic to use in recurrent networks
- ▶ Alternative: **Layer Normalization**
 - the math is same, except statistics is calculated over channels rather than batch elements
 - the effect is quite different though
 - e.g. Layer Normalization “entangles” different neurons within a layer

Image from
<https://arxiv.org/pdf/1803.08494.pdf>



Summary

- ▶ If done wrong, weight initialization may cause the gradients to **vanish or explode**

Summary

- ▶ If done wrong, weight initialization may cause the gradients to **vanish or explode**
- ▶ Neural networks can be regularized with L1/L2 penalties or early stopping

Summary

- ▶ If done wrong, weight initialization may cause the gradients to **vanish or explode**
- ▶ Neural networks can be regularized with L1/L2 penalties or early stopping
- ▶ Dropout makes neurons **create useful features** rather than rely on other neurons to correct their mistakes

Summary

- ▶ If done wrong, weight initialization may cause the gradients to **vanish or explode**
- ▶ Neural networks can be regularized with L1/L2 penalties or early stopping
- ▶ Dropout makes neurons **create useful features** rather than rely on other neurons to correct their mistakes
- ▶ Batch normalization is an **extremely powerful** regularization technique, though the reason for that is not entirely clear

Summary

- ▶ If done wrong, weight initialization may cause the gradients to **vanish or explode**
- ▶ Neural networks can be regularized with L1/L2 penalties or early stopping
- ▶ Dropout makes neurons **create useful features** rather than rely on other neurons to correct their mistakes
- ▶ Batch normalization is an **extremely powerful** regularization technique, though the reason for that is not entirely clear

- ▶ Food for thought: how exactly would you implement an early stopping rule?

Thank you!



amaevskij@hse.ru



SiLiKhon



hse_lambda

Artem Maevskiy