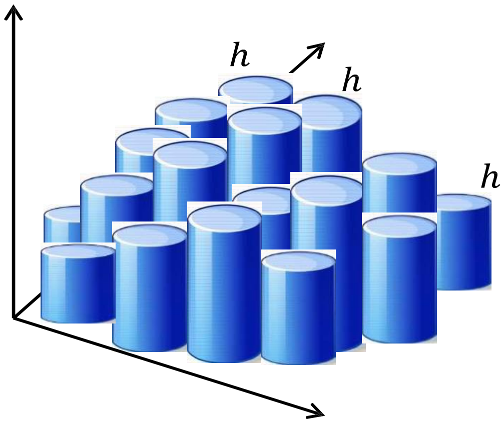
EN.601.482/682 Deep Learning

Training Part I

Activation, Initialization, Preprocessing, Dropout, Batch Norm

Mathias Unberath, PhD
Assistant Professor
Dept of Computer Science
Johns Hopkins University

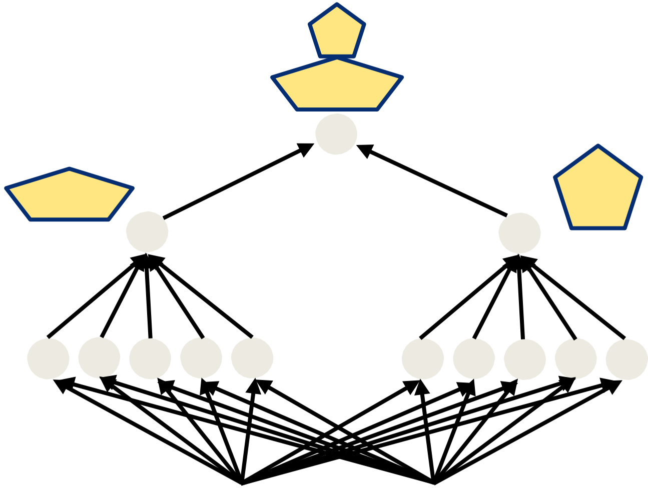
Reminder

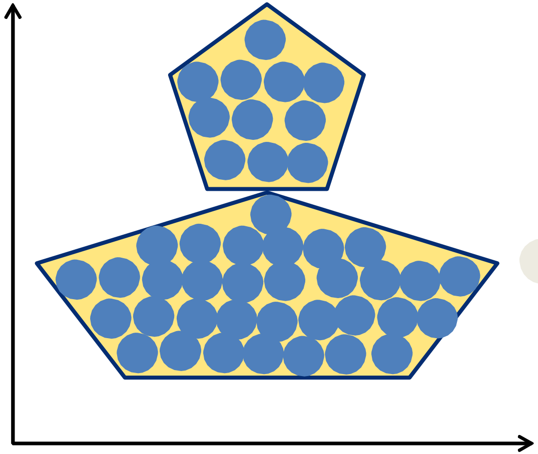


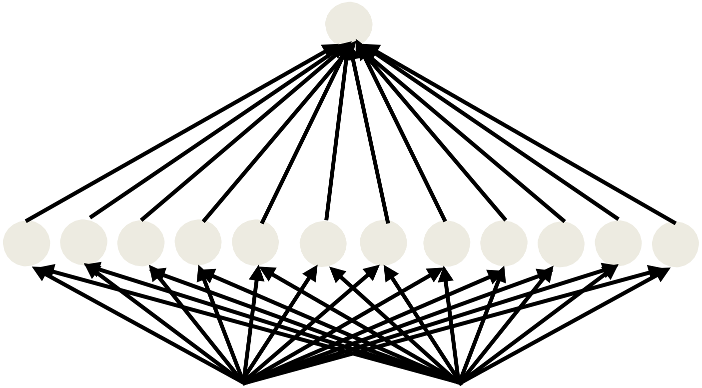
Multi-layer perceptrons are

- Universal Boolean functions
- Universal Classifiers
- Universal approximators

… but may require infinitely many neurons





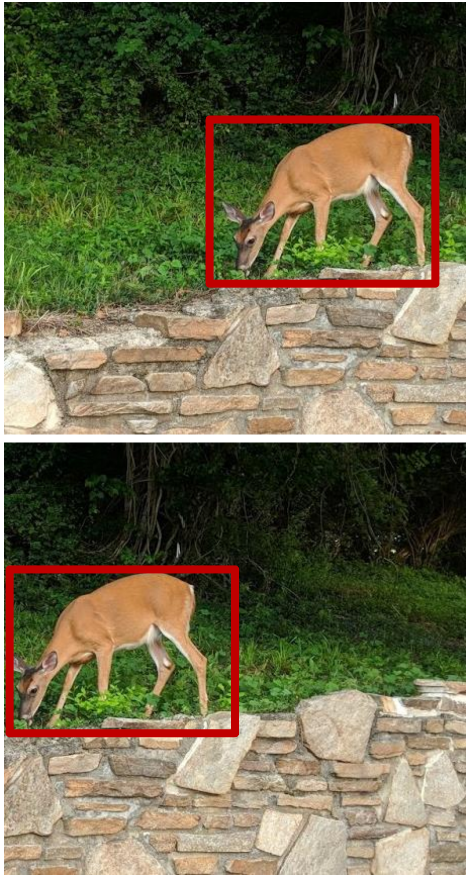


Reminder

- MLPs to recognize patterns:
  Weights act as **templates**

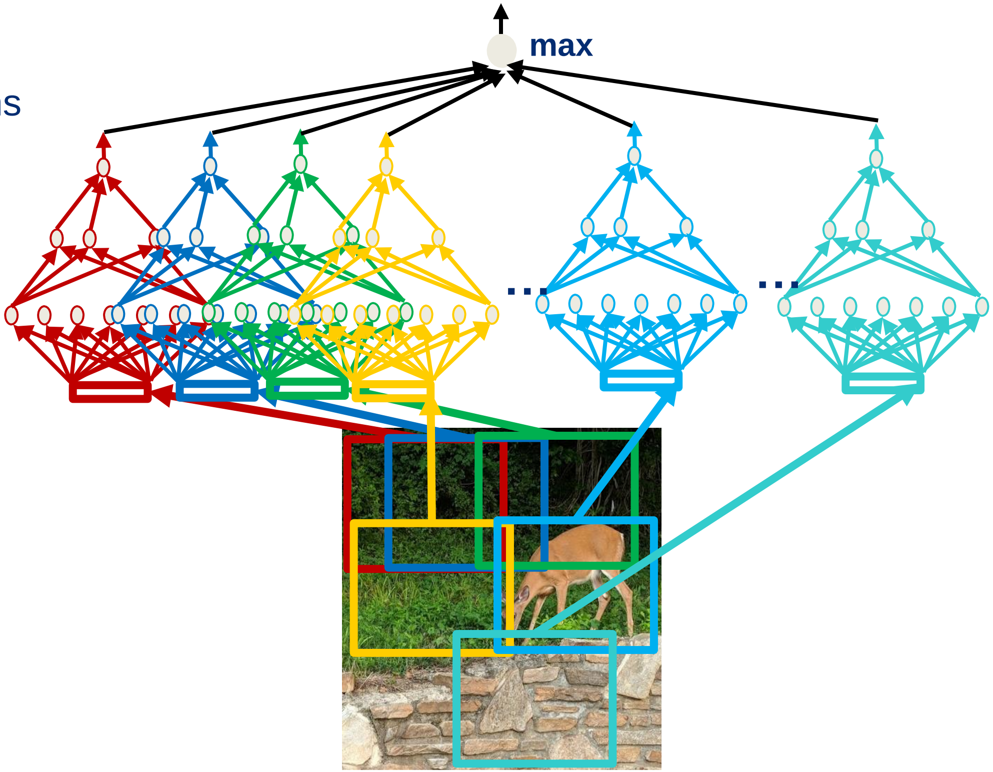
- MLPs are **not shift invariant**

- For many problems, however:
  → Location of pattern does not matter



Reminder

- First idea
  - Apply multiple MLPs at different locations
  - Max over outputs
  - Large number of parameters

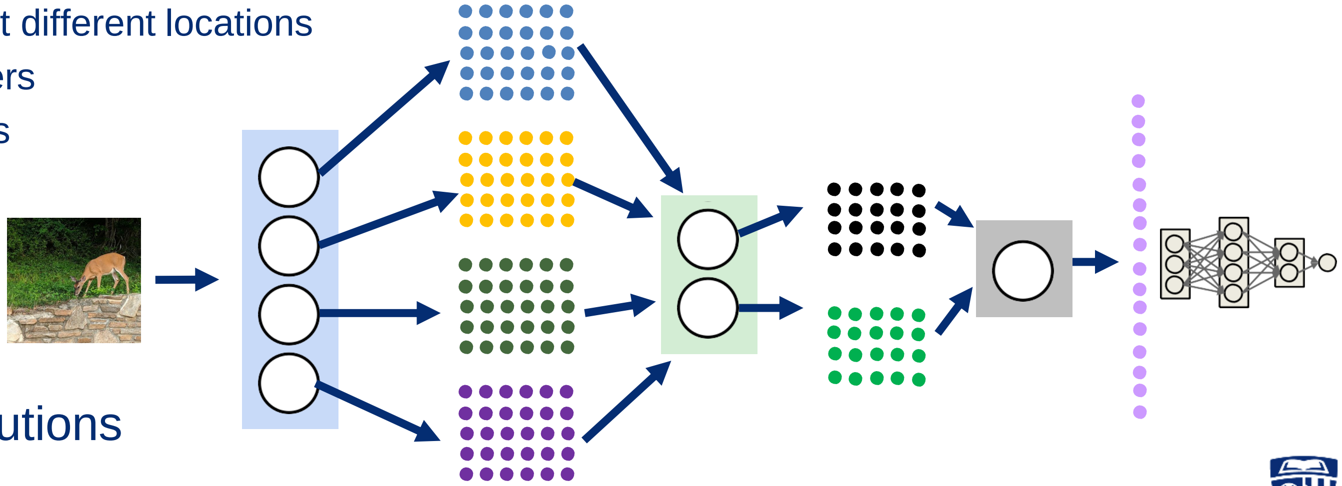


Reminder

- First idea
  - Apply multiple MLPs at different locations
  - Max over outputs
  - Large number of parameters

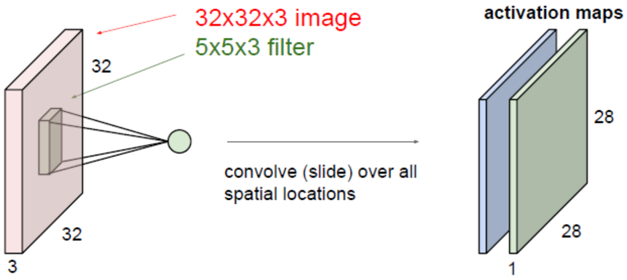
- Second idea
  - Apply same MLP at different locations
  - Reduced parameters
  - Distributed analysis

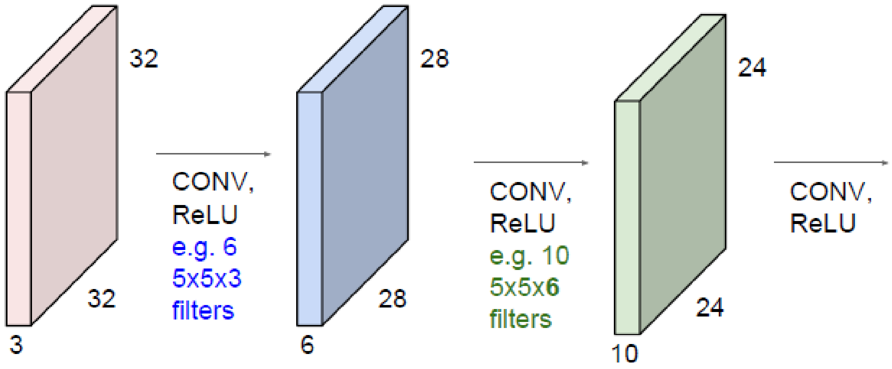
Lower layers learn local features
Deeper layers learn abstract features



→ Realized as convolutions

Reminder





Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Reminder

- Convolutional Neural Networks (ConvNets) stack
  - Convolutional layers
  - Pooling layers
  - Fully connected (FC) layers

- Typical architecture:
  [(Conv→Activation)*N → Pool]*M → (FC→Activation)*F, Softmax

Today (and next time): **Considerations important for training**

- Design choices
- Potential pitfalls

Today’s Lecture

Connecting the Dots

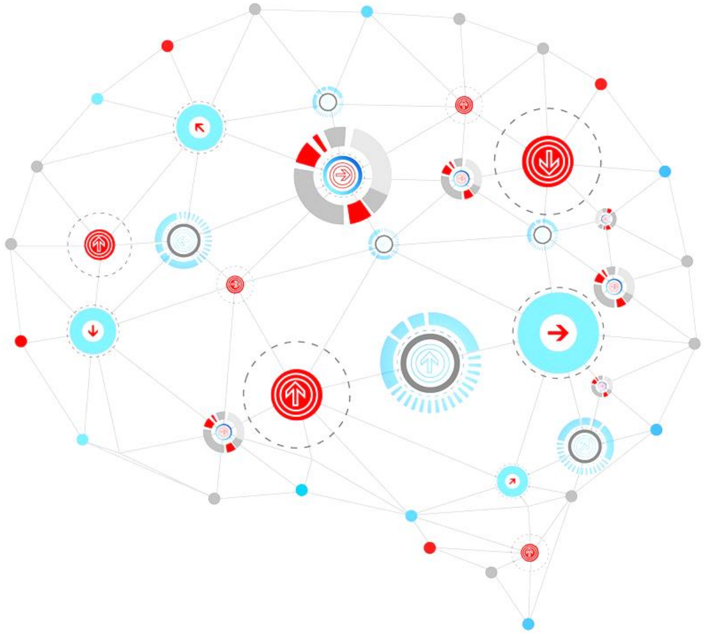
Activation

Initialization

Preprocessing

Dropout

Batch norm



Activation, Initialization, Preprocessing, Dropout, Batch Norm

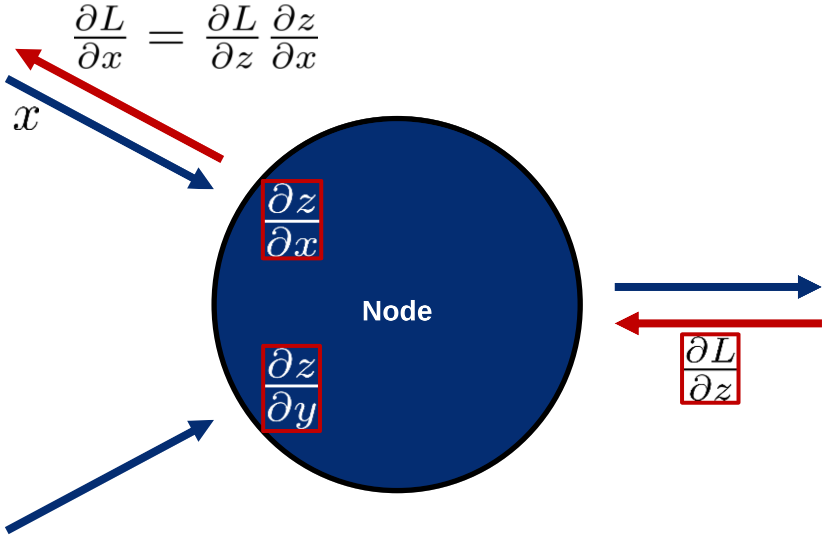
Connecting the Dots

Training MLPs and ConvNets

- Neural networks are universal approximators
  → **But we must train them to approximate a function!**

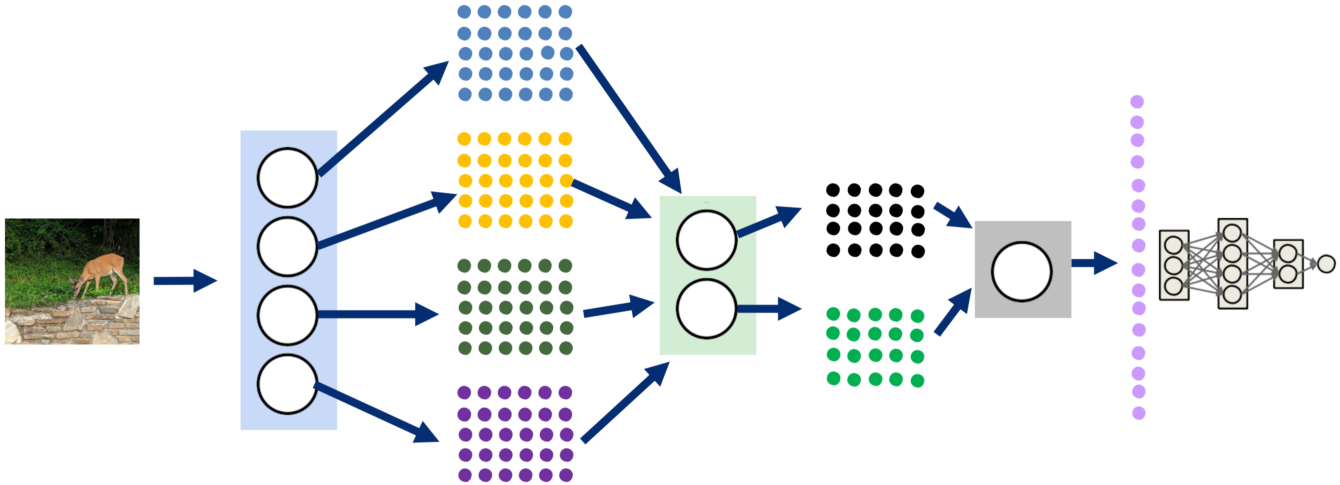
- Computational graphs
  - Consecutively apply chain rule
  - Analytic gradient computation for arbitrarily complex functions

- Convolutional neural networks
  - Same concept still applies!
  - More complicated with shared weights



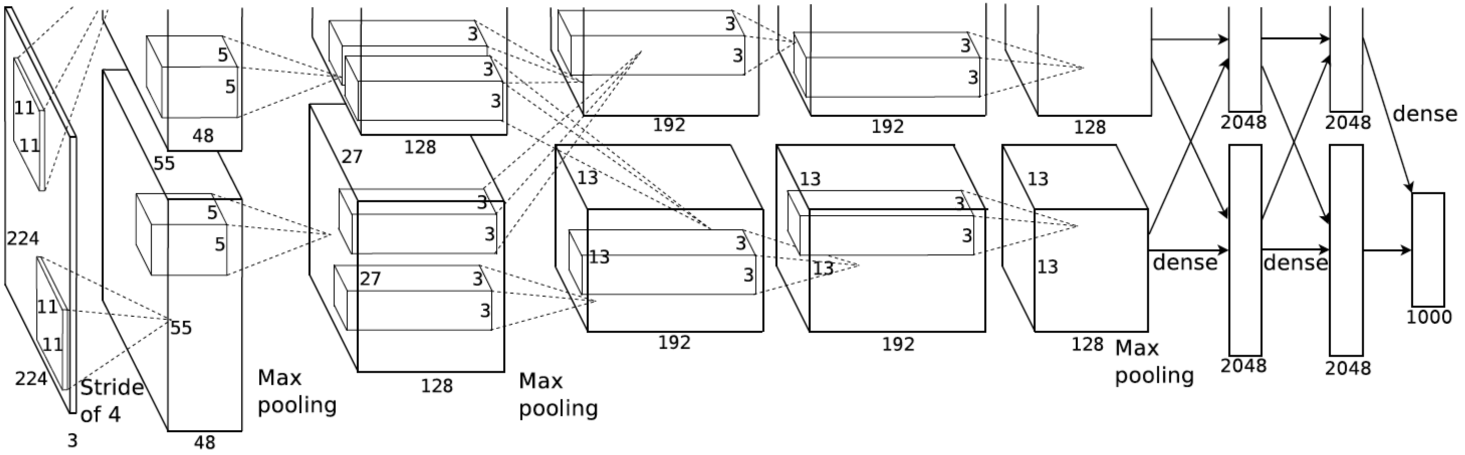
Training MLPs and ConvNets

- This is just a computational graph



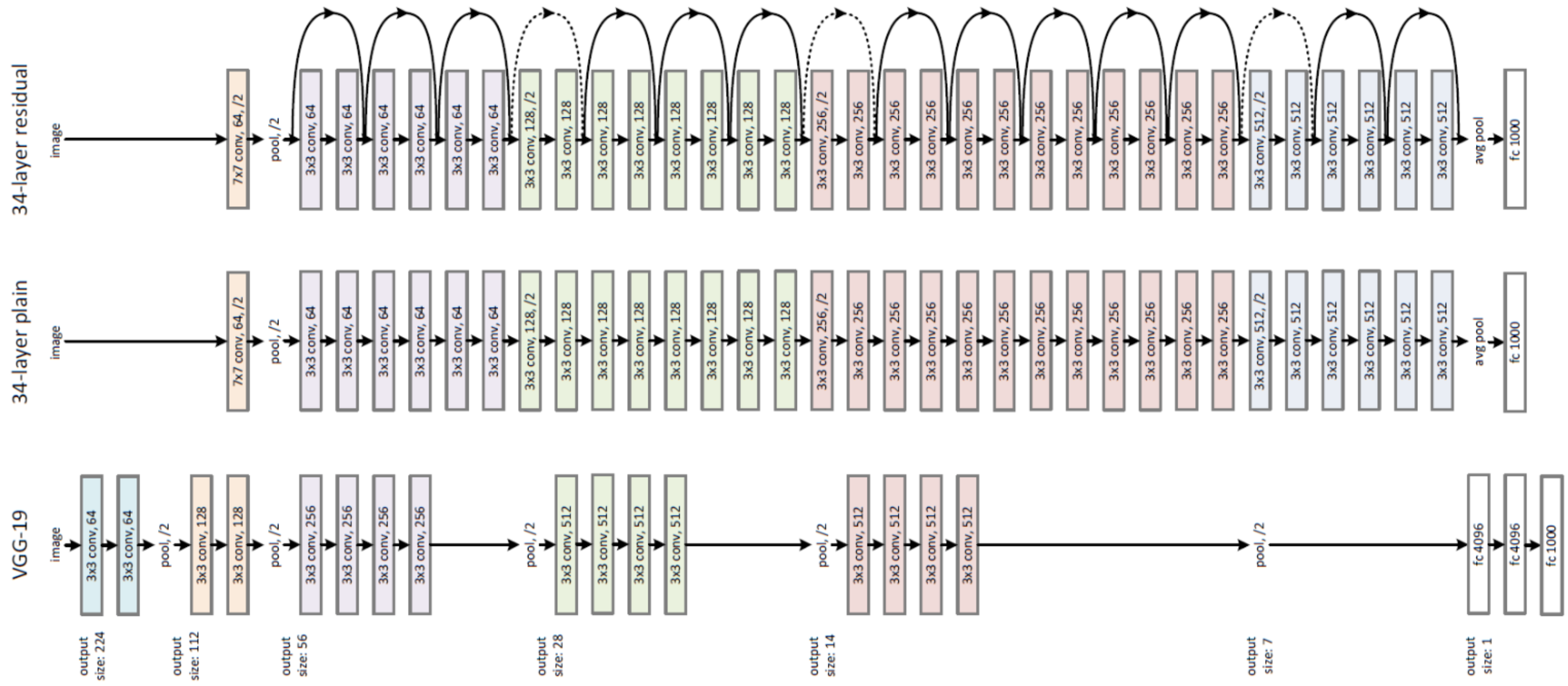
Training MLPs and ConvNets

- And so is this



Training MLPs and ConvNets

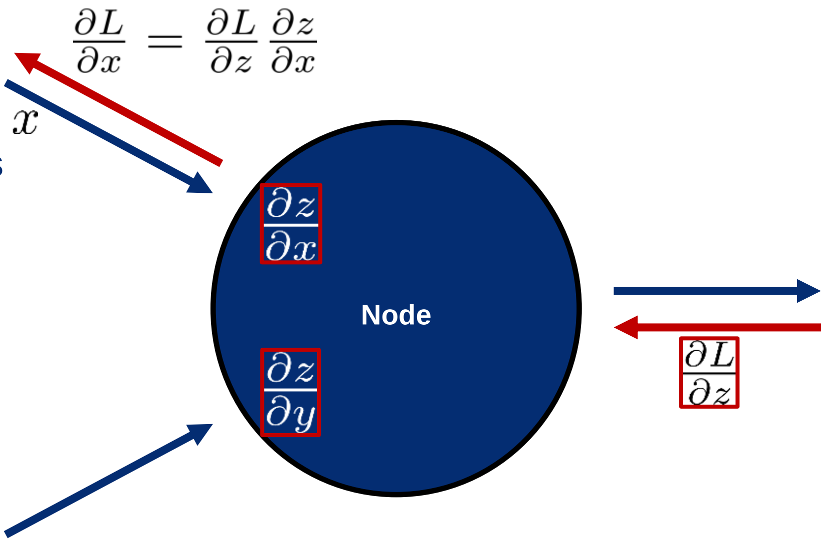
- And these



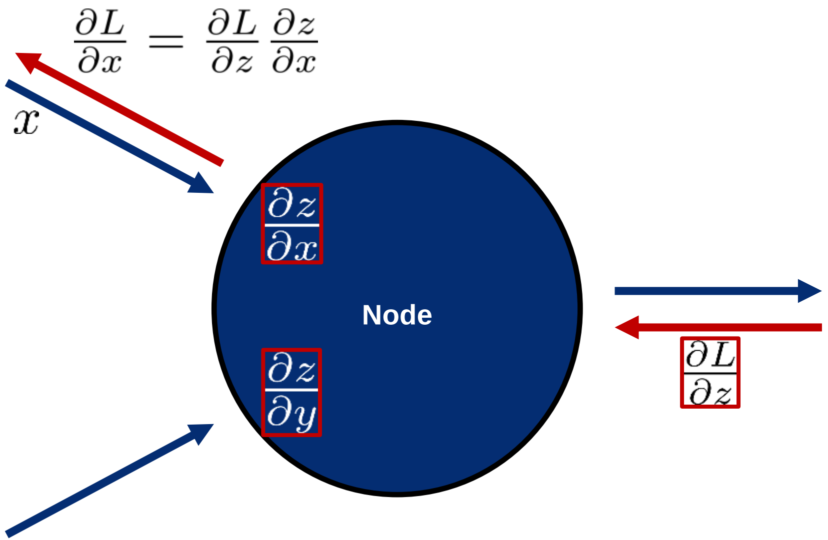
Training MLPs and ConvNets

- **One time setup:** Architecture, Loss, etc.
  - Define forward/backward for every node
  - Setup computational graph by connecting nodes
  - Obtain training/validation/test data

- Apply batched SGD
  → Until not converged, do:
  - **Sample** batch of data
  - **Forward prop** through graph, compute loss
  - **Backward prop** to compute gradients
  - **Update** parameters using gradient



Training MLPs and ConvNets



Today

→ Focus on components relevant for setup

→ Design considerations

→ We will assume the perfect architecture has been found
Selecting it will be discussed a bit later…

Activation, Initialization, Preprocessing, Dropout, Batch Norm

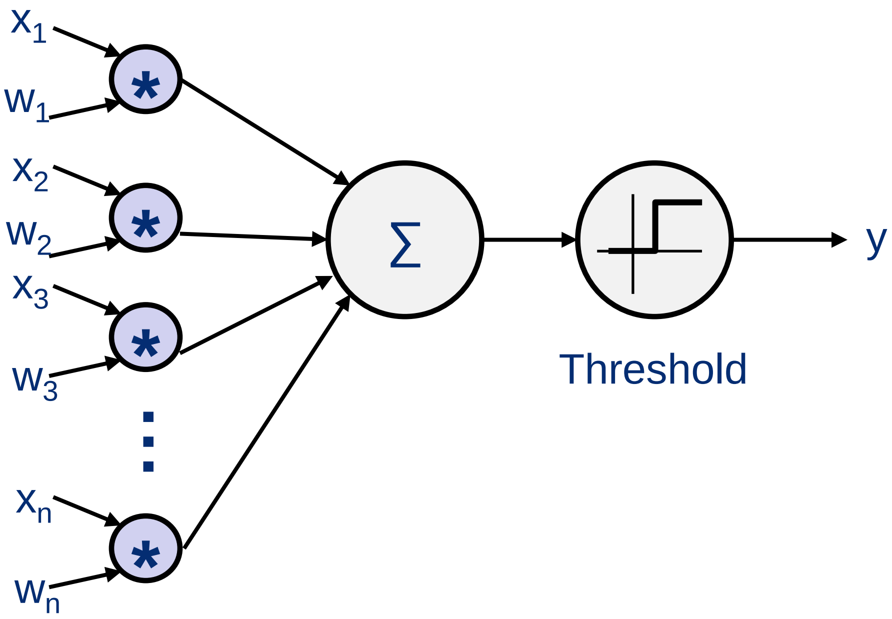
Activation

The Neural View of Perceptrons

The perceptron

- Weighted sum of inputs
- Non-linearity: If exceeds threshold

→ Unit fires!



Recap from 3 slides ago:

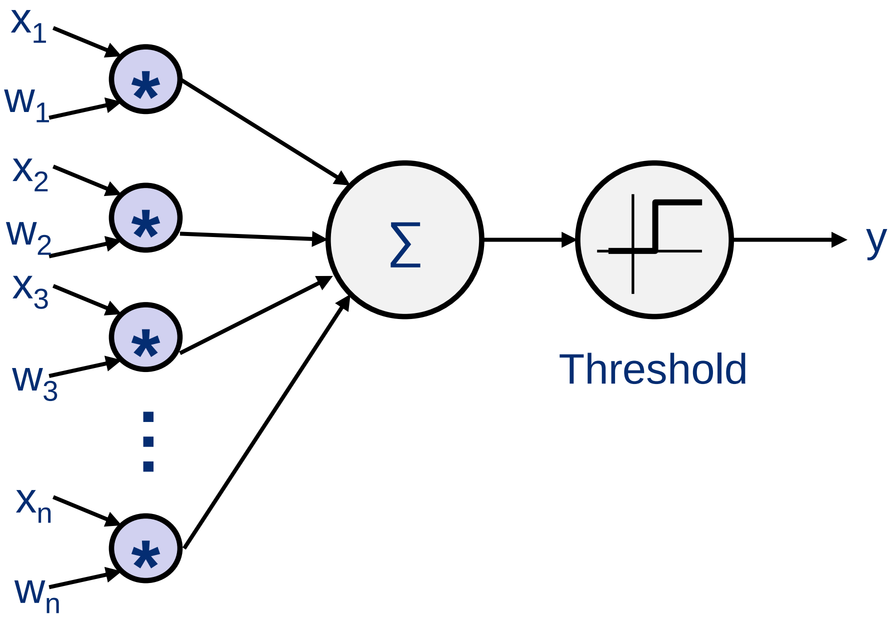
How do we train neural networks or even perceptrons in practice?

The Neural View of Perceptrons

The perceptron

- Weighted sum of inputs
- Non-linearity: If exceeds threshold

→ Unit fires!



Recap from 3 slides ago:

How do we train neural networks or even perceptrons in practice?

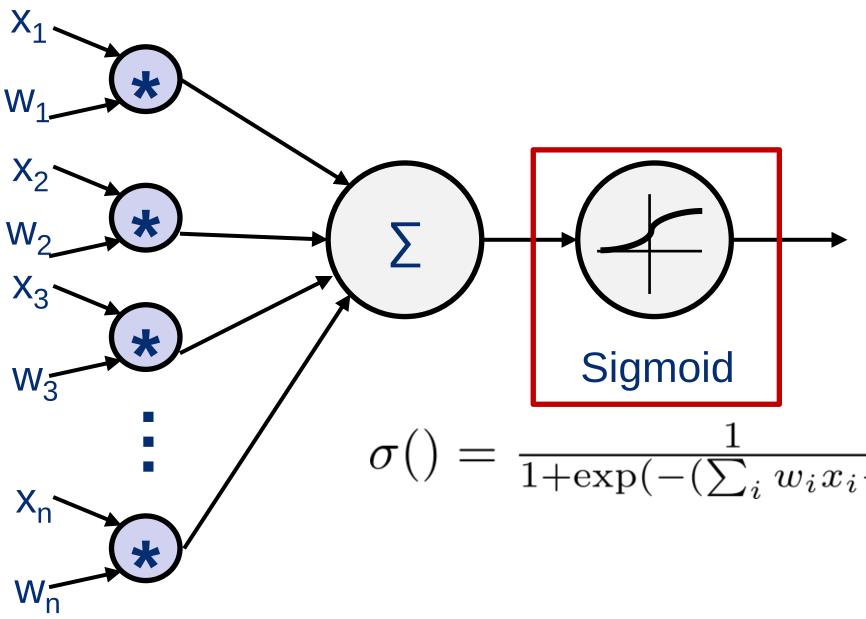
→ Gradient descent

The Neural View of Perceptrons

The perceptron

- Weighted sum of inputs
- Non-linearity: If exceeds threshold

→ Unit fires!



$$\sigma() = \frac{1}{1+\exp(-(\sum_i w_i x_i + b))}$$

Recap from 3 slides ago:

How do we train neural networks or even perceptrons in practice?

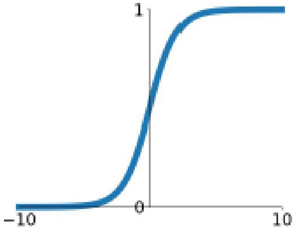
→ Gradient descent

→ Introduce differentiable functions

The Zoo of Common Activation Functions

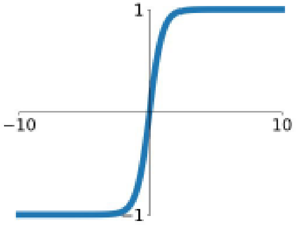
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



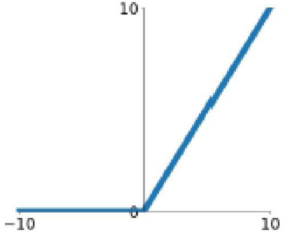
tanh

$\tanh(x)$



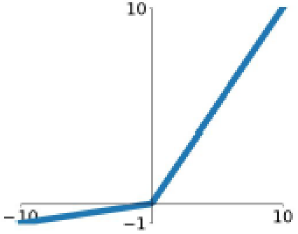
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

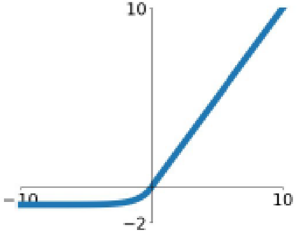


Maxout

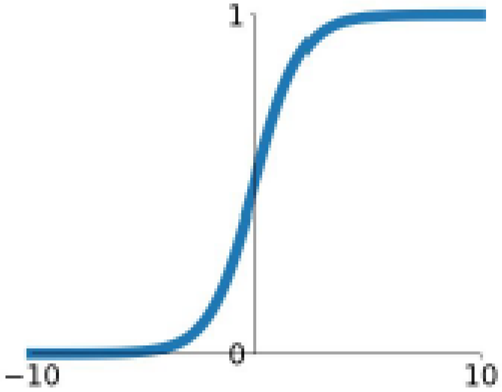
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



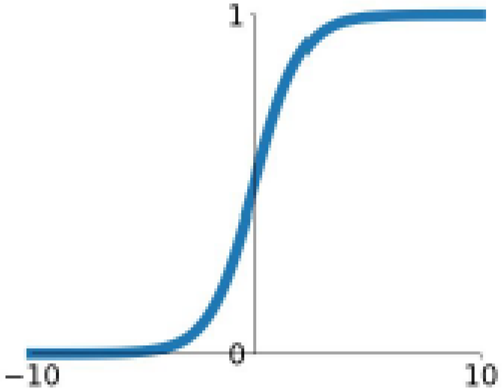
Sigmoid



Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$

- Squashes input to [0,1]
- Historically popular:
  Saturating firing rate of a neuron

Sigmoid



Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$

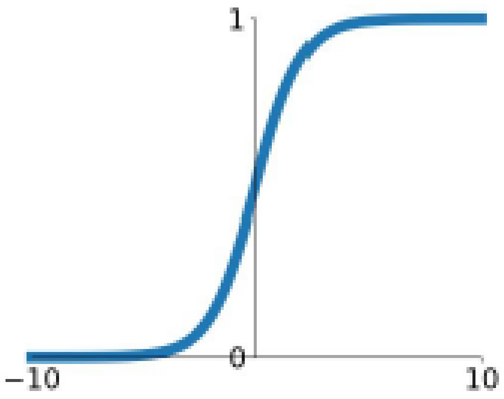
- Squashes input to [0,1]
- Historically popular:
  Saturating firing rate of a neuron

Problems

- Gradient: $\frac{\partial \sigma(x)}{\partial x} = (1 - \sigma(x))\sigma(x)$
  → Gradient vanishes for saturated neurons

As an example: What is the gradient at x = -10, x = 0, x = 10?

Sigmoid



Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$

- Squashes input to [0,1]
- Historically popular:
  Saturating firing rate of a neuron

Problems

- Gradient: $\frac{\partial \sigma(x)}{\partial x} = (1 - \sigma(x))\sigma(x)$
  → Gradient vanishes for saturated neurons
- Outputs are not zero-centered

Sigmoid

What happens if all inputs are positive?

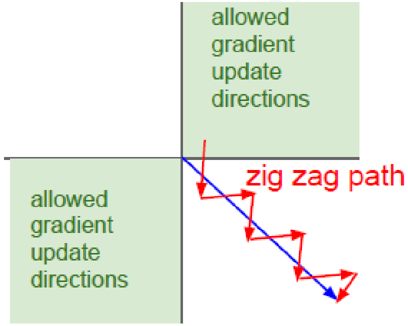
$$f(\sum_i w_i x_i + b)$$

Gradients on w?

Local gradient is x → all positive!

Upstream gradient is pos. or neg.

→ Gradient is either all pos. or all neg.!



→ Ineffective gradient updates!

Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$

- Squashes input to [0,1]
- Historically popular: Saturating firing rate of a neuron

Problems

- Gradient: $\frac{\partial \sigma(x)}{\partial x} = (1 - \sigma(x))\sigma(x)$
  → Gradient vanishes for saturated neurons
- Outputs are not zero-centered

Sigmoid

This is also why we want zero-mean data!

What happens if all inputs are positive?

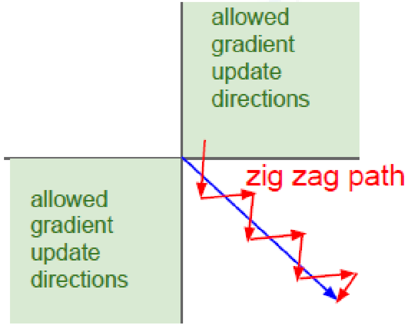
$$f(\sum_i w_i x_i + b)$$

Gradients on w?

Local gradient is *x* → all positive!

Upstream gradient is pos. or neg.

→ Gradient is either all pos. or all neg.!



→ Ineffective gradient updates!

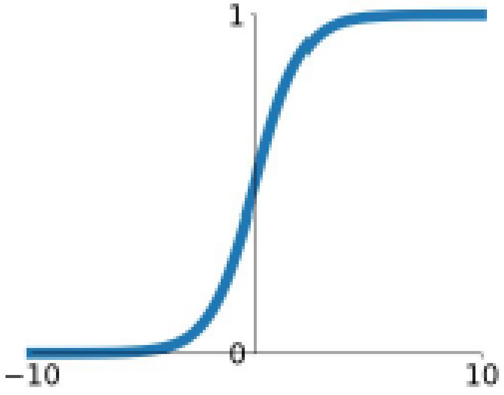
Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$

- Squashes input to [0,1]
- Historically popular:
  Saturating firing rate of a neuron

Problems

- Gradient: $\frac{\partial \sigma(x)}{\partial x} = (1 - \sigma(x))\sigma(x)$
  → Gradient vanishes for saturated neurons
- Outputs are not zero-centered

Sigmoid



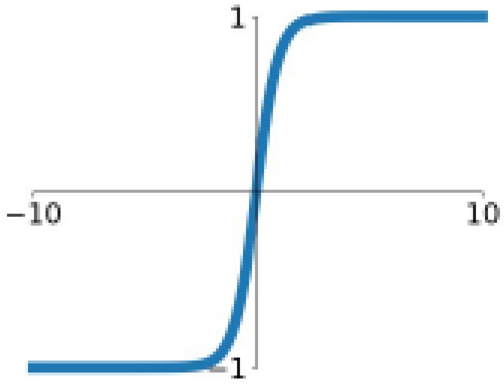
Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$

- Squashes input to [0,1]
- Historically popular:
  Saturating firing rate of a neuron

Problems

- Gradient: $\frac{\partial \sigma(x)}{\partial x} = (1 - \sigma(x))\sigma(x)$
  → Gradient vanishes for saturated neurons
- Outputs are not zero-centered
- exp() computation is a bit expensive

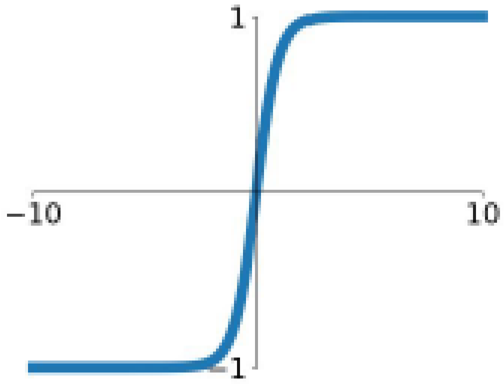
Tanh



Tanh

- Squashes input to [-1,1]
- Zero-centered output

Hyperbolic Tangent



Tanh

- Squashes input to [-1,1]
- Zero-centered output

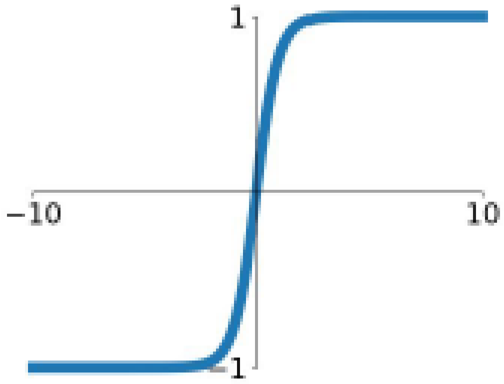
Problems

- Gradient vanishes for saturated neurons

“Recommended tanh”: $f(x) = 1.7159 \tanh(2/3\ x)$ (LeCun 1991)

LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K. R. (2012). Efficient backprop. In Neural networks: Tricks of the trade (pp. 9-48). Springer, Berlin, Heidelberg.

Hyperbolic Tangent



Tanh

- Squashes input to [-1,1]
- Zero-centered output

Problems

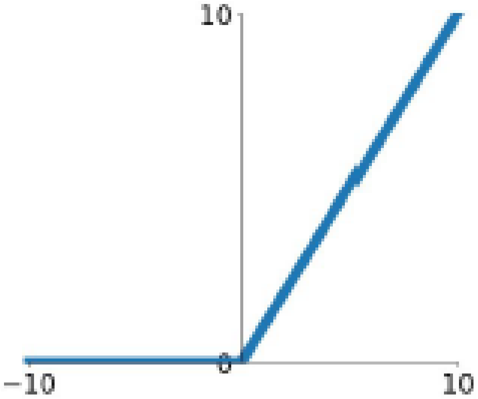
- Gradient vanishes for saturated neurons

"Recommended tanh": $f(x) = 1.7159 \tanh(2/3\ x)$ (LeCun 1991)

Read this! If not now, then when starting projects at the latest!

LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K. R. (2012). Efficient backprop. In Neural networks: Tricks of the trade (pp. 9-48). Springer, Berlin, Heidelberg.

Rectified Linear Unit

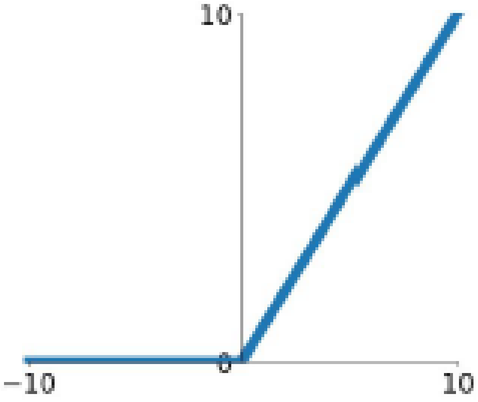


ReLU $\mathrm{ReLU}(x) = \max(0, x)$

- No saturation in positive regime
- Computationally efficient
- Converges much faster than previous func.s
- Closer to biological neuron activation

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. NeurIPS (pp. 1097-1105).

Rectified Linear Unit



ReLU $\mathrm{ReLU}(x) = \max(0, x)$

- No saturation in positive regime
- Computationally efficient
- Converges much faster than previous func.s
- Closer to biological neuron activation

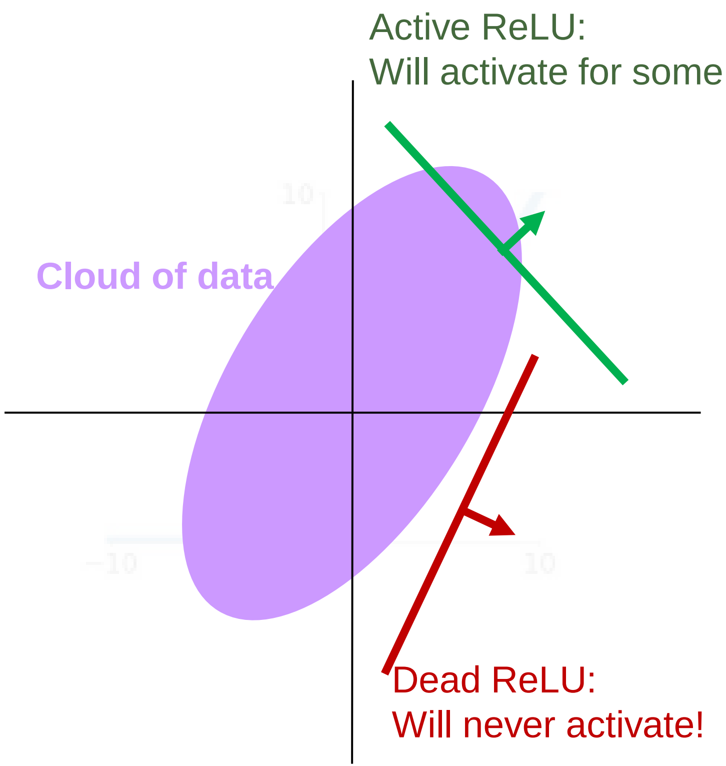
Problems

- Again not zero-centered!

As an example: What is the gradient at x = -10, x = 0, x = 10?

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. NeurIPS (pp. 1097-1105).

Rectified Linear Unit



ReLU $\mathrm{ReLU}(x) = \max(0, x)$

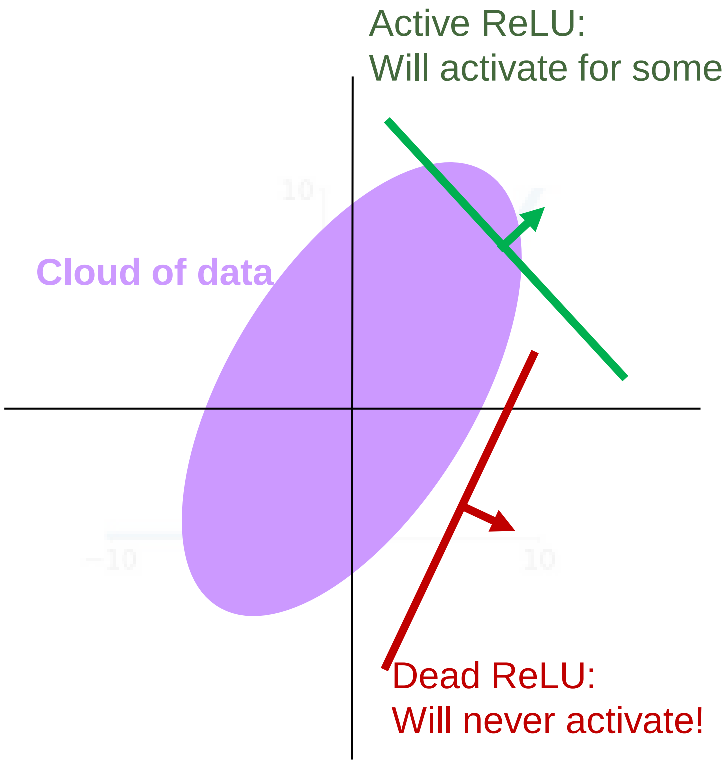
- No saturation in positive regime
- Computationally efficient
- Converges much faster than previous func.s
- Closer to biological neuron activation

Problems

- Again not zero-centered!

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. NeurIPS (pp. 1097-1105).

Rectified Linear Unit



ReLU $\mathrm{ReLU}(x) = \max(0, x)$

- No saturation in positive regime
- Computationally efficient
- Converges much faster than previous func.s
- Closer to biological neuron activation

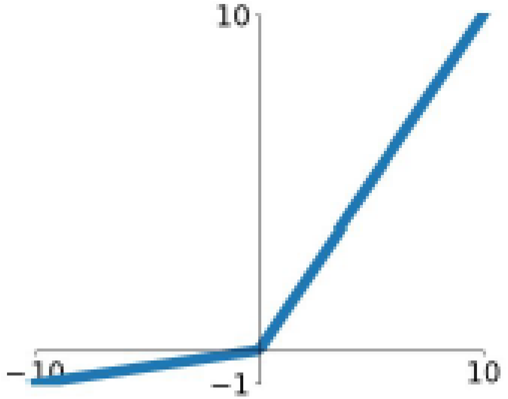
Problems

- Again not zero-centered!

→ Bias term to the rescue!
Initialize with small positive bias

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. NeurIPS (pp. 1097-1105).

Leaky Rectified Linear Unit

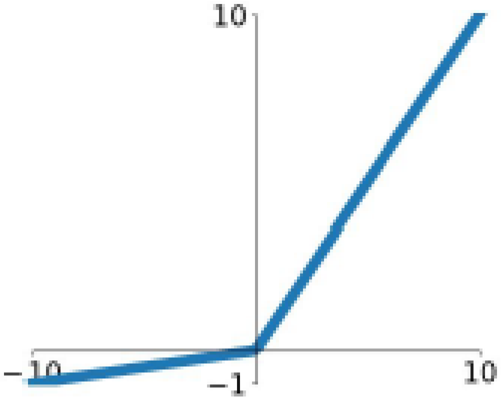


Leaky ReLU $\mathrm{ReLU}(x) = \max(\alpha x, x)$

- No saturation
- Computationally efficient
- Converges much faster than previous func.s
- Will **not die**

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. CVPR.

Leaky Rectified Linear Unit



Leaky ReLU $\mathrm{ReLU}(x) = \max(\alpha x, x)$

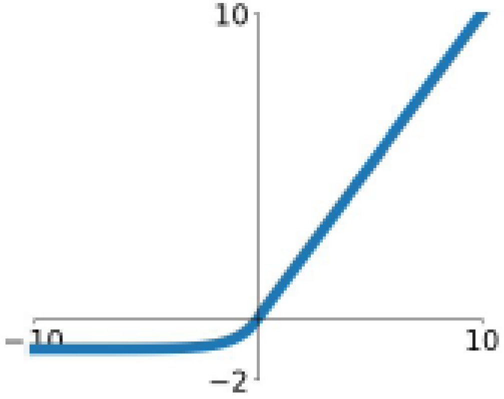
- No saturation
- Computationally efficient
- Converges much faster than previous func.s
- Will **not die**

Leaky ReLU:
α is constant and small, e.g. 0.01

Parametric Rectifier PReLU:
Backpropagation into α!

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. CVPR.

Exponential Linear Unit



ELU

$$\mathrm{ELU}(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1), & \text{if } x < 0 \end{cases}$$

- Benefits of ReLU
- Closer to zero mean
- Saturates in negative regime
  → "Noise-robust deactivation state"

Problems

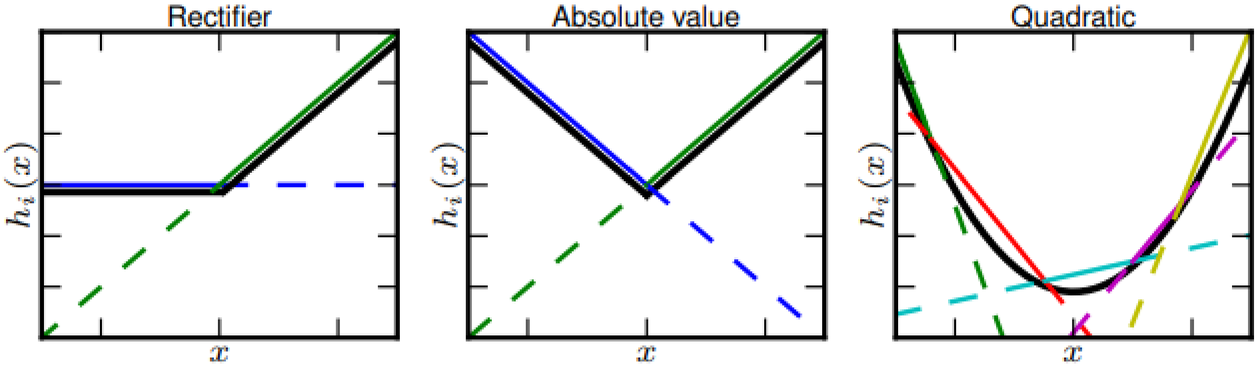
- exp() is a bit expensive to compute

Clevert, D. A., Unterthiner, T., & Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). arXiv:1511.07289.

Maxout

$$\max(W_1 x + b_1, W_2 x + b_2)$$

- Does not follow the conventional “dot product/convolution → activation”
- Generalizes ReLU etc.
- Linear regime: Does not die!



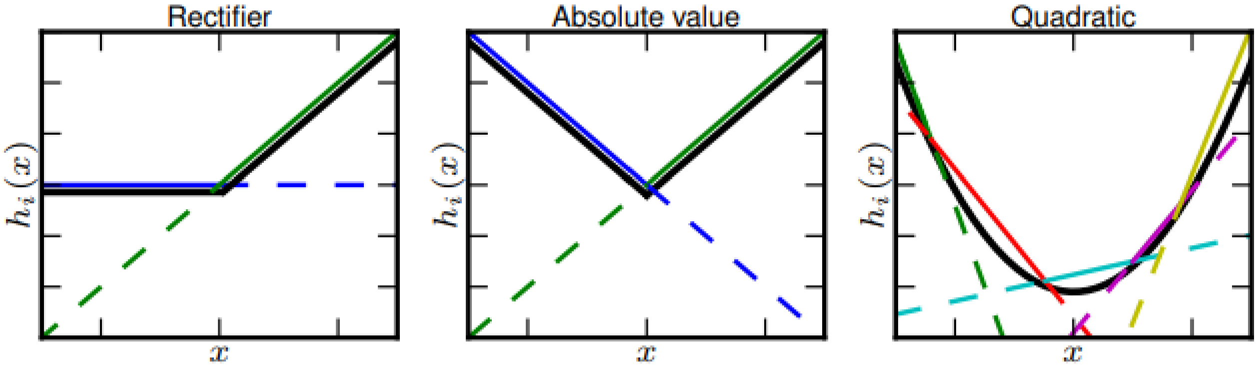
Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., & Bengio, Y. (2013). Maxout networks. arXiv preprint arXiv:1302.4389.

Maxout

$$\max(W_1 x + b_1, W_2 x + b_2)$$

- Does not follow the conventional “dot product/convolution → activation”
- Generalizes ReLU etc.
- Linear regime: Does not die!

- **Doubles the number of parameters per maxout neuron**



Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., & Bengio, Y. (2013). Maxout networks. arXiv preprint arXiv:1302.4389.

Recap and Take Away (if nothing else)

- Use ReLU!
- Try Leaky ReLU, PReLU, ELU, maybe even maxout

- Try tanh if you have time
- Do **not** use sigmoid

→ Be careful with learning rates!

Activation, Initialization, Preprocessing, Dropout, Batch Norm

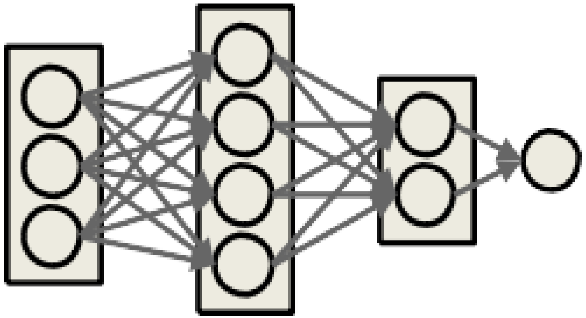
Initialization

Weight Initialization

Where are we now?

- Architecture is decided (number of neurons, activation functions)
- Close to start training!

But: Where should we start? How do we initialize our weights/parameters?



Weight Initialization

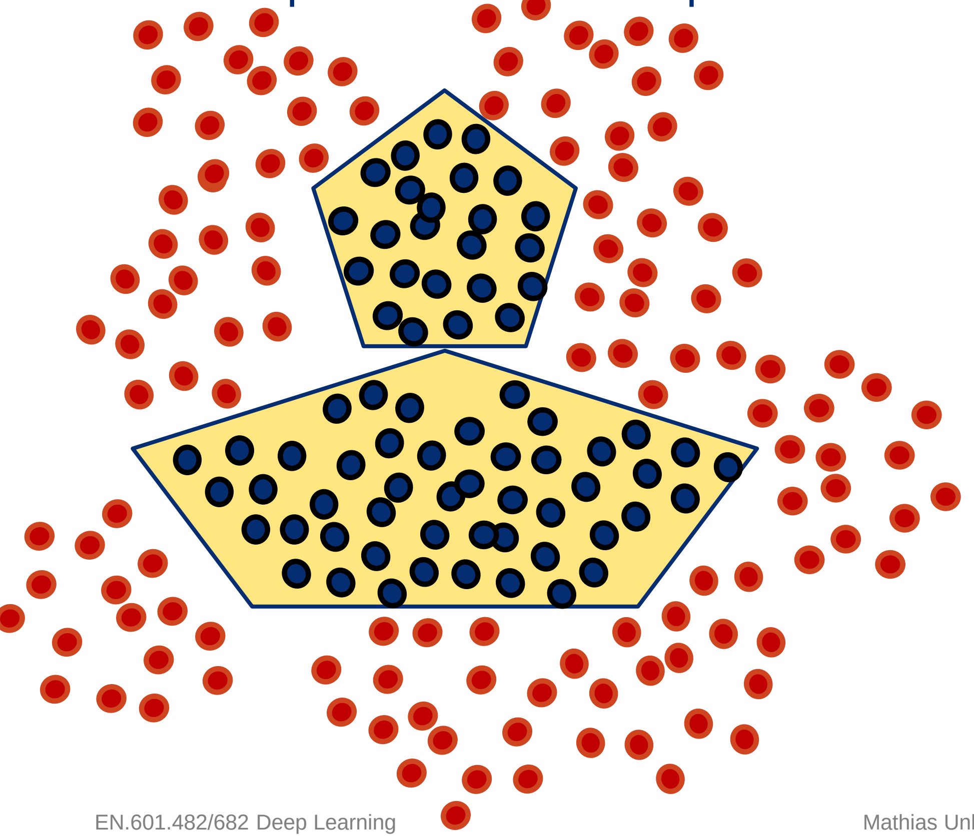
First Idea:

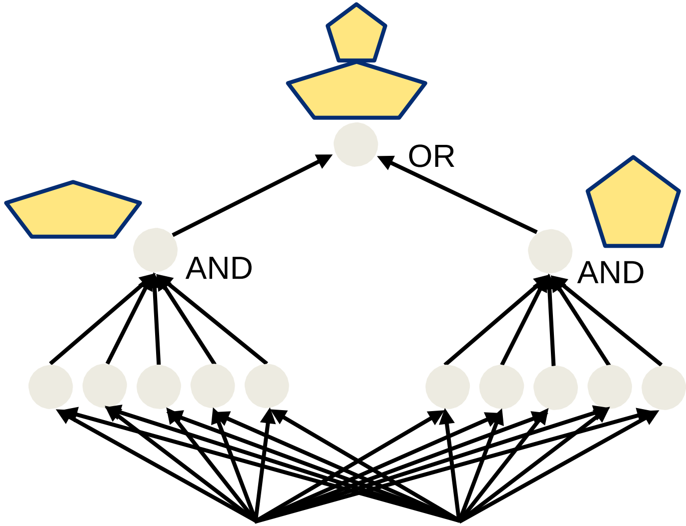
To make training more reproducible, initialize every weight with a constant.
Because 0 is a nice number (and zero-centering is a thing) we use 0.

Q: Why is this a bad idea?

Weight Initialization

An attempt at a visual example

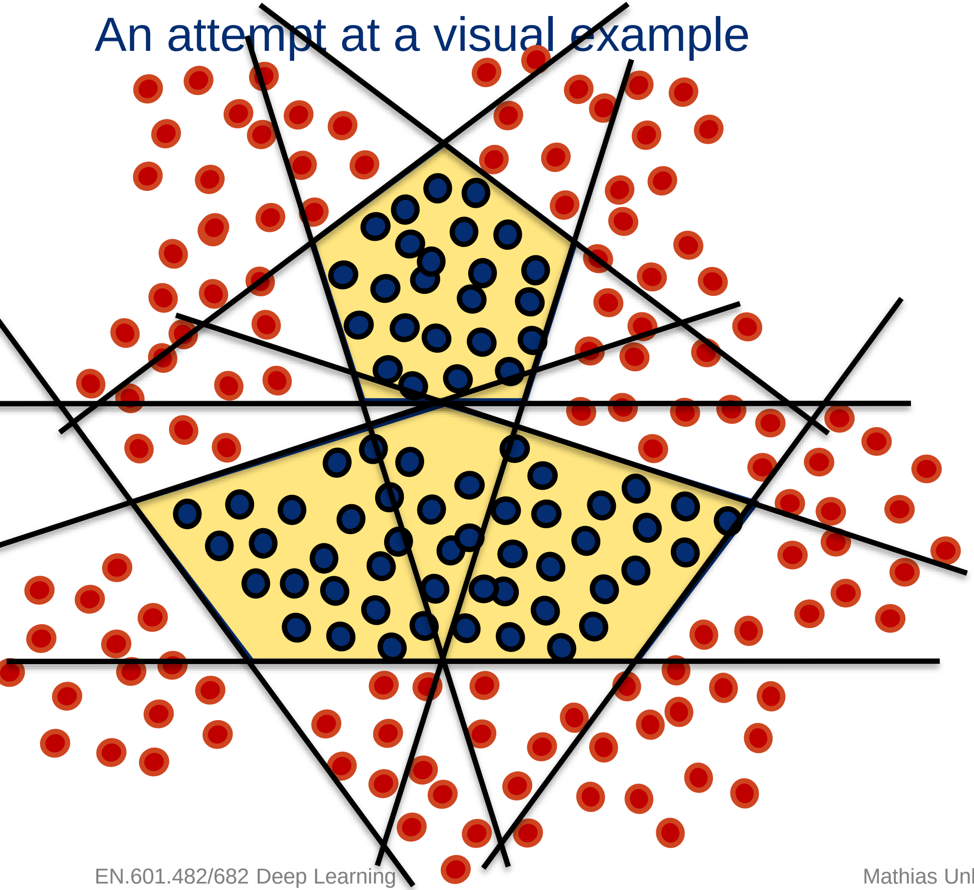


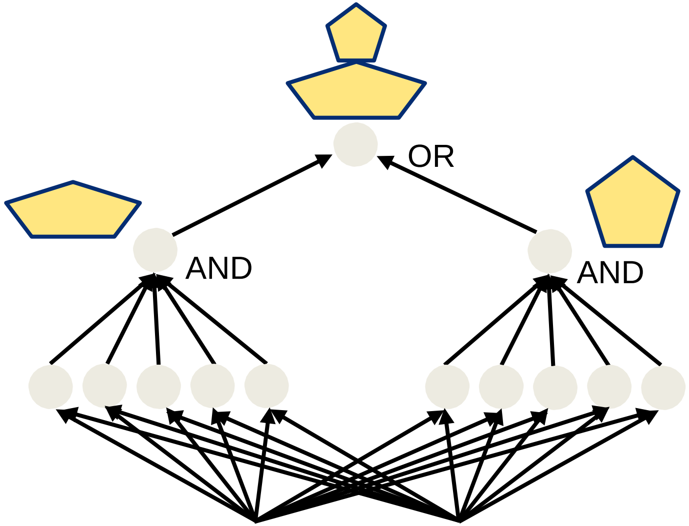


This network can perfectly describe the required decision boundary!

Weight Initialization

An attempt at a visual example



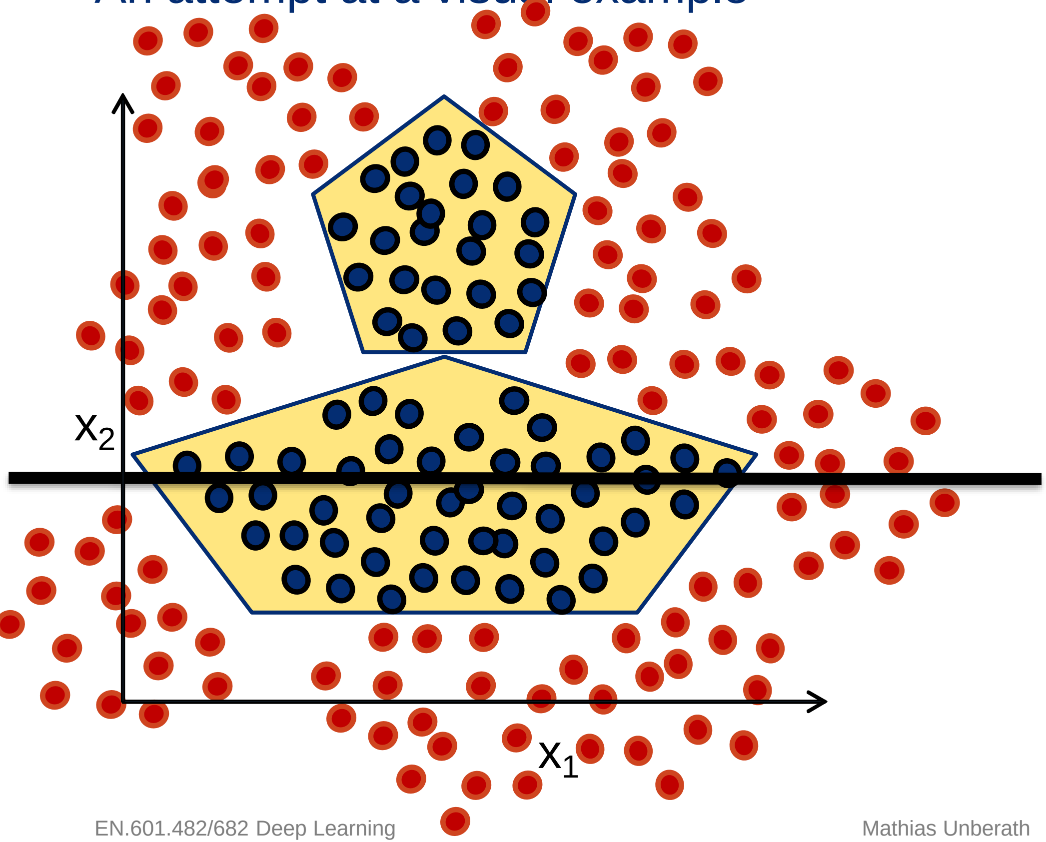


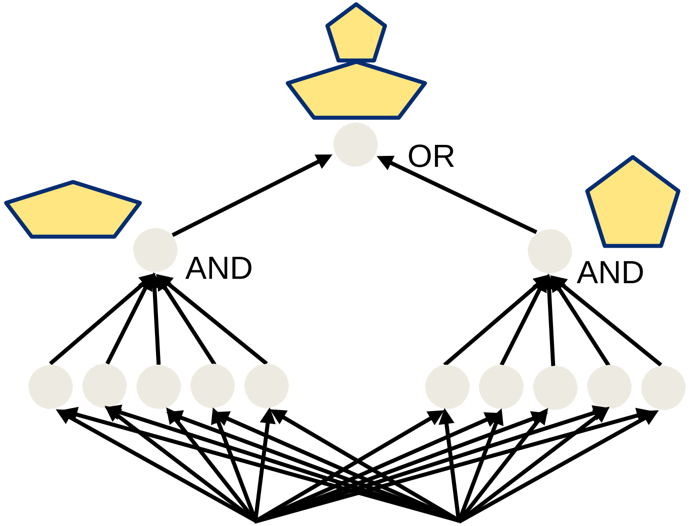
This network can perfectly describe the required decision boundary!

These are the lines a network must discover during training

Weight Initialization

An attempt at a visual example



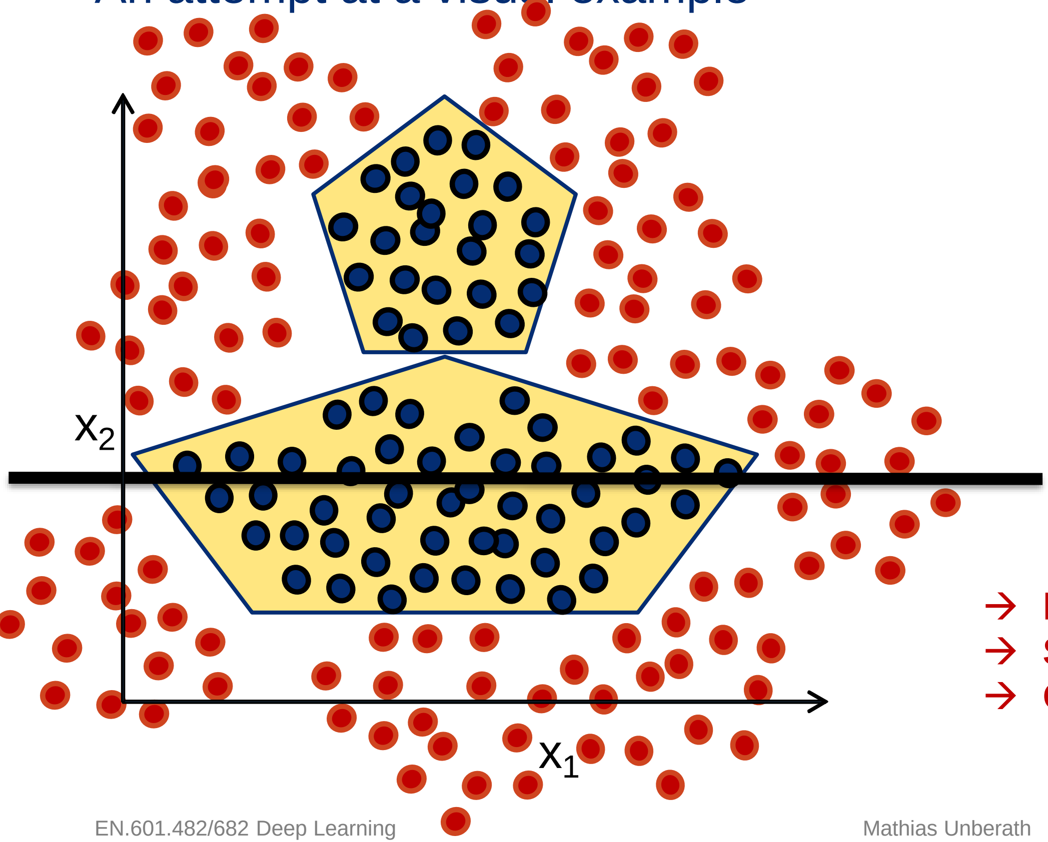


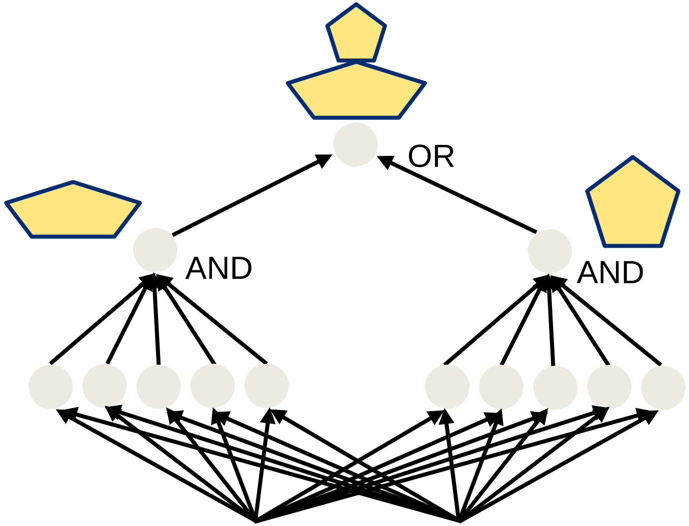
This network can perfectly describe the required decision boundary!

Constant initialization:
e.g. $\mathbf{(0,c)^T(x1,x2) + 0}$

Weight Initialization

An attempt at a visual example



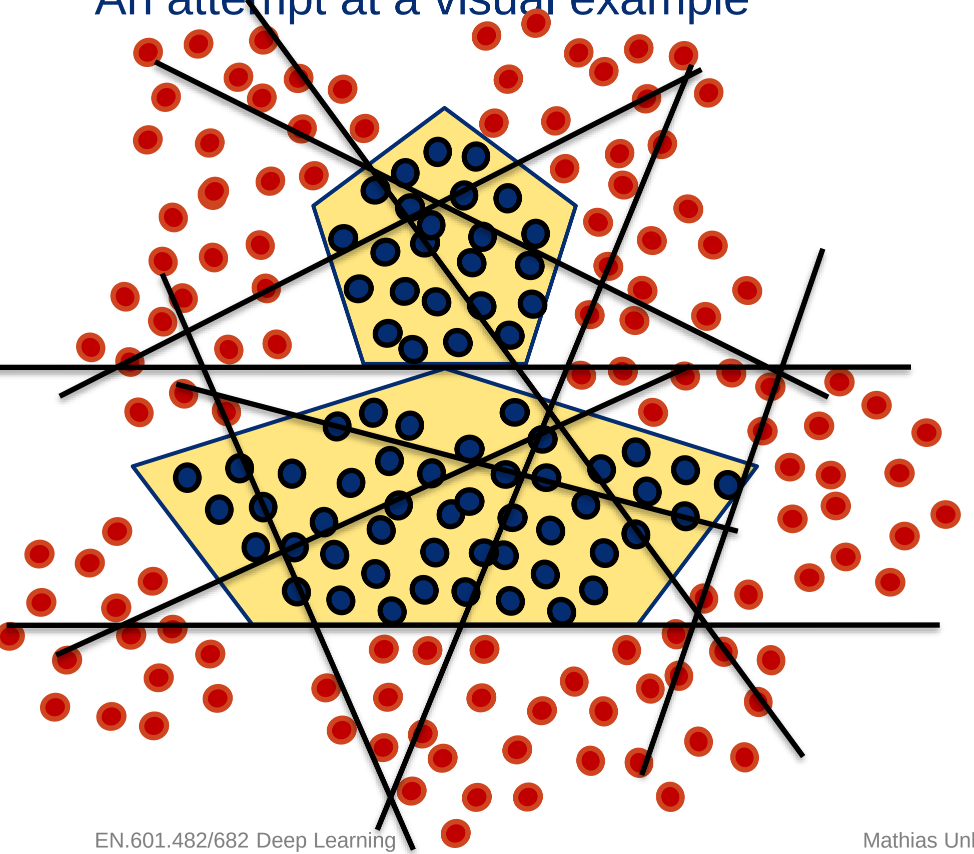


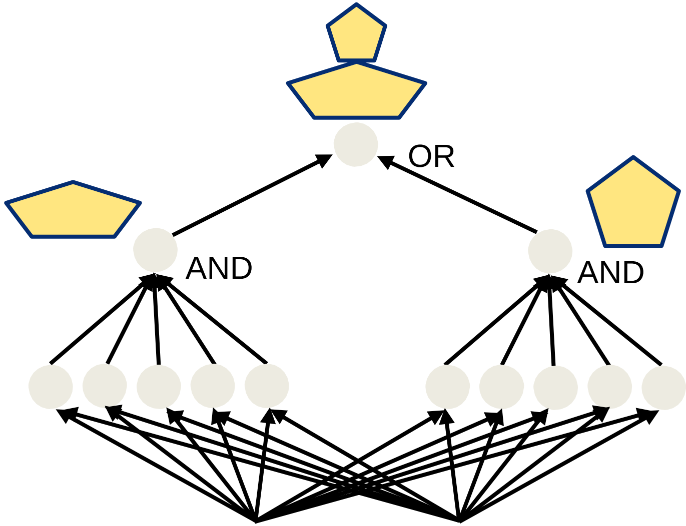
This network can perfectly describe the required decision boundary!

- → **Every neuron "sees" the same situation!**
- → **System is entirely symmetric**
- → **Gradient/updates are the same for every neuron!**

Weight Initialization

An attempt at a visual example



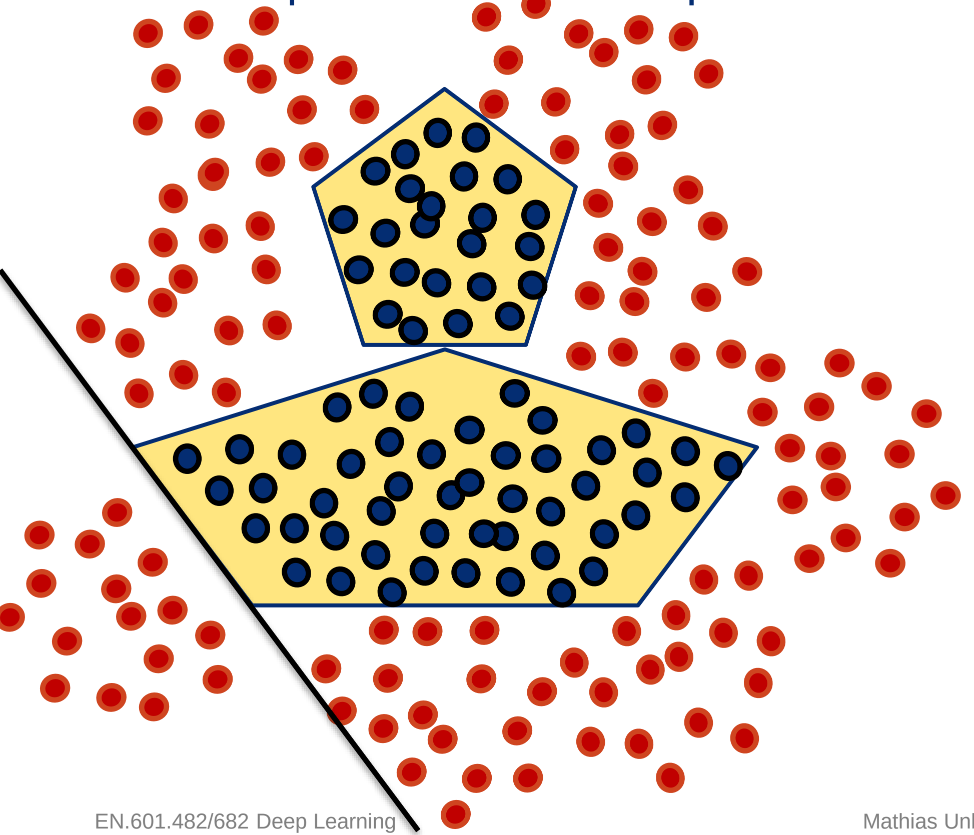


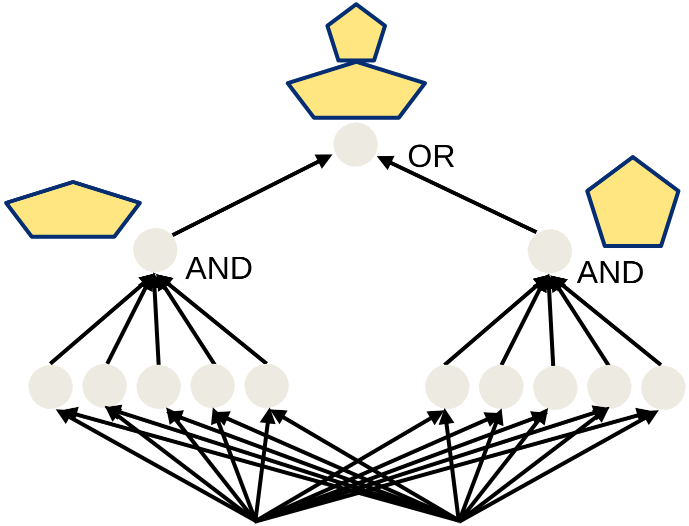
This network can perfectly describe the required decision boundary!

→ **Random initialization**

Another Aside

An attempt at a visual example



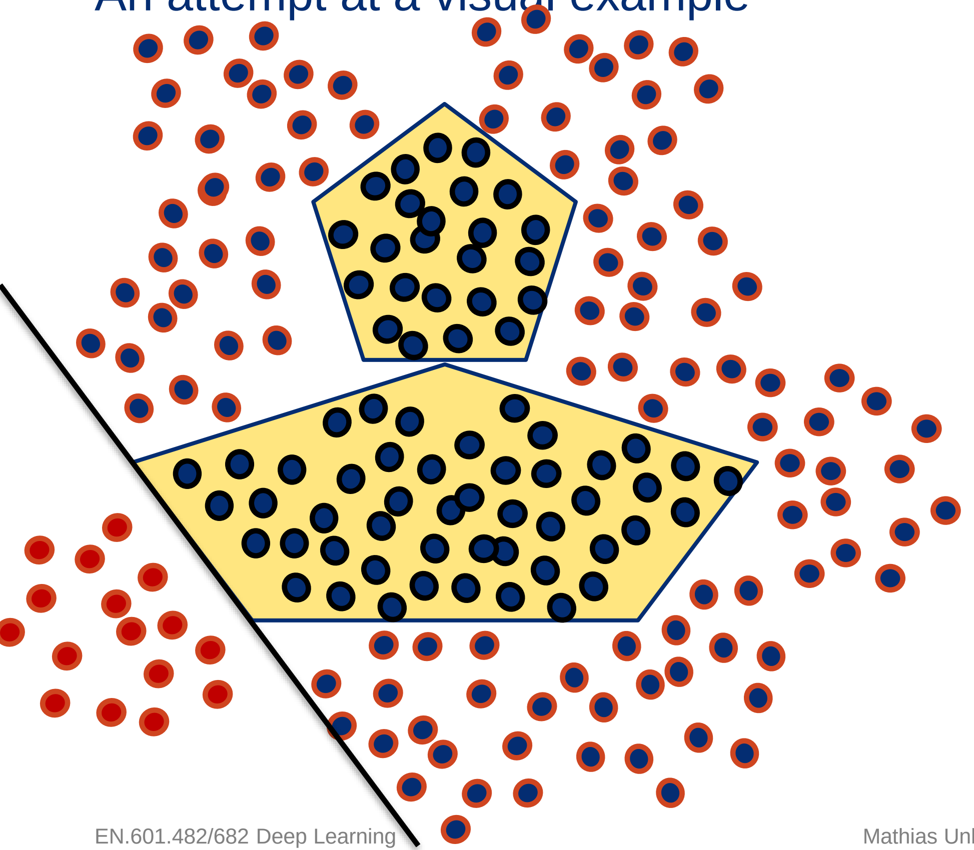


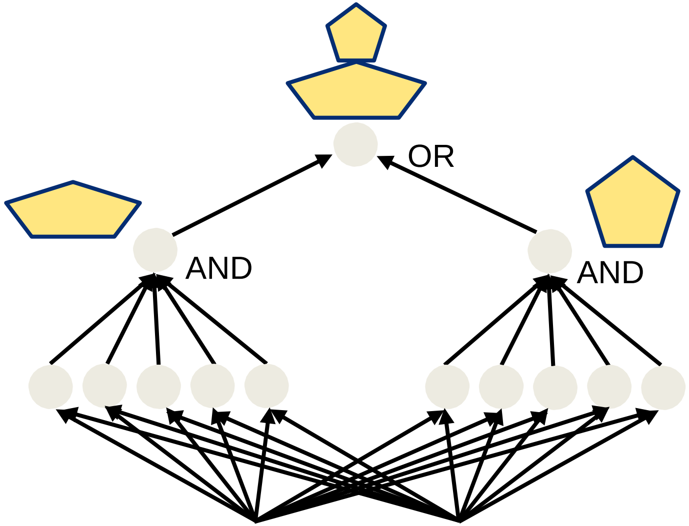
This network can perfectly describe the required decision boundary!

Q: During training, we want to find this line in one of the shallow layers.
Why is it difficult to find this line?

Another Aside

An attempt at a visual example





~~This network can perfectly describe the required decision boundary!~~

Only in theory, because in practice it is very unlikely to converge to this solution.
Network must have larger capacity!

Weight Initialization

Second Idea:
Small random numbers with zero mean and some standard deviation

```
W = 0.01* np.random.randn(D,H)
```

Works OK with shallow networks but is problematic with deeper ones.

Weight Initialization

Third Idea: Xavier initialization
Small random numbers with zero mean and well-defined standard deviation

```
W = np.random.randn(fan_in, fan_out) * np.sqrt(fan_in) # layer initialization
```

Works well, but breaks with ReLU. Why?

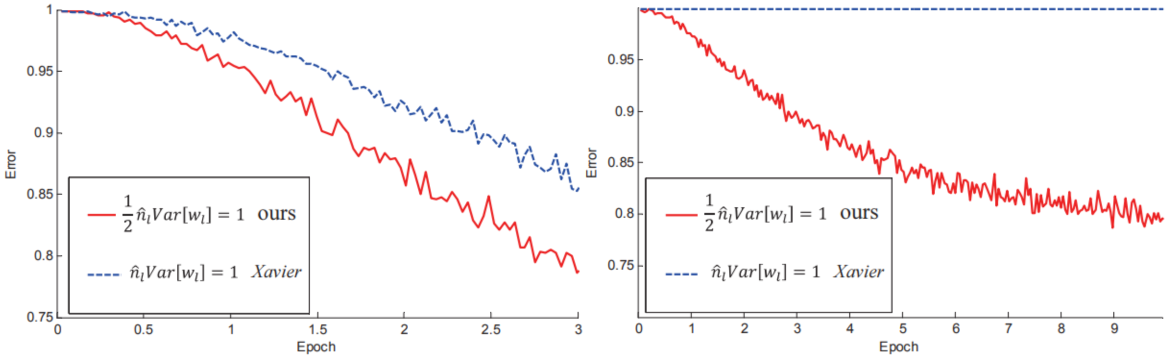
Because derivation is based on linear neuron assumption. After Xavier initialization, outputs will be in the ~linear regime for tanh and sigmoid, but obviously not for ReLU.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. Artificial intelligence and statistics (pp. 249-256).

Weight Initialization

Fourth Idea: He initialization
Small random numbers with zero mean and well-defined standard deviation

```
W = np.random.randn(fan_in, fan_out) * np.sqrt(2/fan_in) # layer initialization
```



He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. ICCV (pp. 1026-1034).

Recap and Take Away (if nothing else)

- Initialization is an active field of research
  (in neural networks and beyond, e.g. image registration)

- Xavier and He initialization played an important role in the success of DL

- If you are using ReLU as recommended: He initialization is your friend!

Activation, Initialization, Preprocessing, Dropout, Batch Norm

Preprocessing

Reminder: The Sigmoid or ReLU Problem

What happens if all inputs are positive?

$f(\sum_i w_i x_i + b)$

Q: Why is this a problem again?

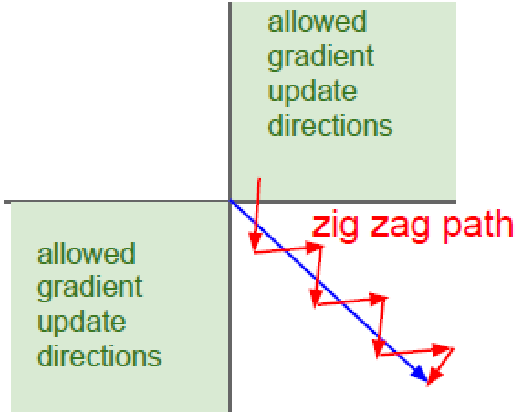
Gradients on w?

Local gradient is x → all positive!

Upstream gradient is positive or negative

→ Gradient is either all positive or all negative!

→ Ineffective updates!



Reminder: The Sigmoid or ReLU Problem

What happens if all inputs are positive?

$f(\sum_i w_i x_i + b)$

Q: Why is this a problem again?

A: Because “normal” images are in [0,255].

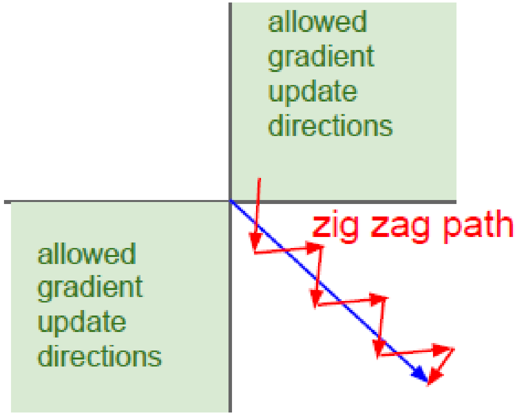
Gradients on w?

Local gradient is x → all positive!

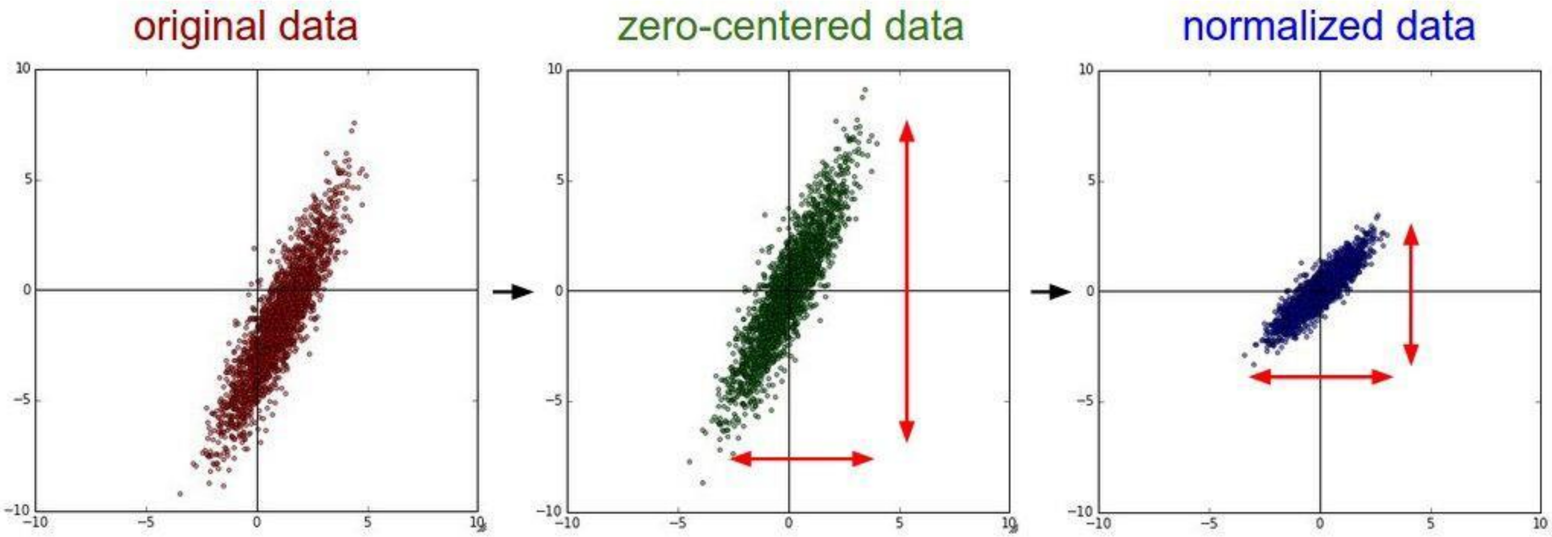
Upstream gradient is positive or negative

→ Gradient is either all positive or all negative!

→ Ineffective updates!



Preprocessing

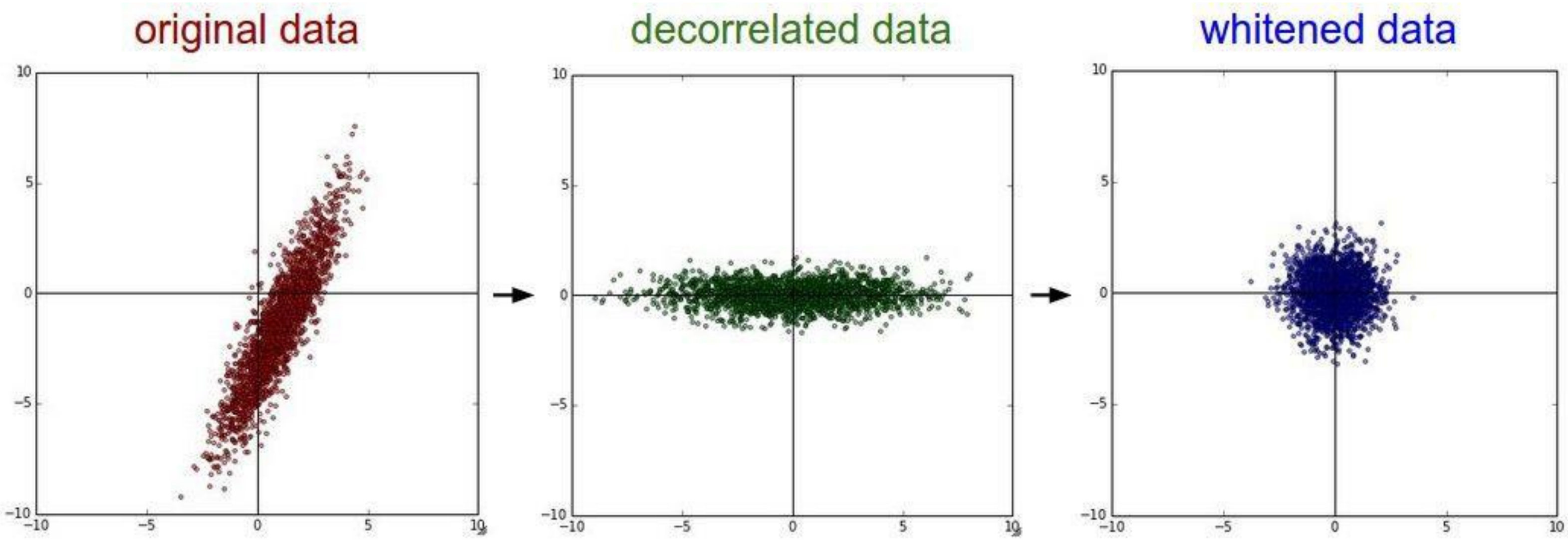


Input data | Mean subtracted | Divided by standard dev

For images, mean centering can be sufficient → Normalization not necessary

Image from Stanford's cs231n.

Preprocessing



Input data

Projected onto PCA

Divided by variance

Whitening: Projection onto axes of highest variation, then normalization
→ Not usually done for images

Image from Stanford's cs231n.

Recap and Take Away (if nothing else)

- Zero-center data
- Try normalizing images

- Do not (necessarily) consider decorrelation, whitening or other techniques for images, but this may be different for other input data

At inference time:

Apply the same transformation (e.g. mean subtraction) with values extracted from the training data.

A Brief (but Important?) Aside

The dynamic range of images

- Natural images (e.g. CIFAR10, ImageNet) are usually 8-bit per channel
  Why? Tradeoff between human perception / storage (8-bit → 255 is nice)

Too coarse quantization is not nice!

A Brief (but Important?) Aside

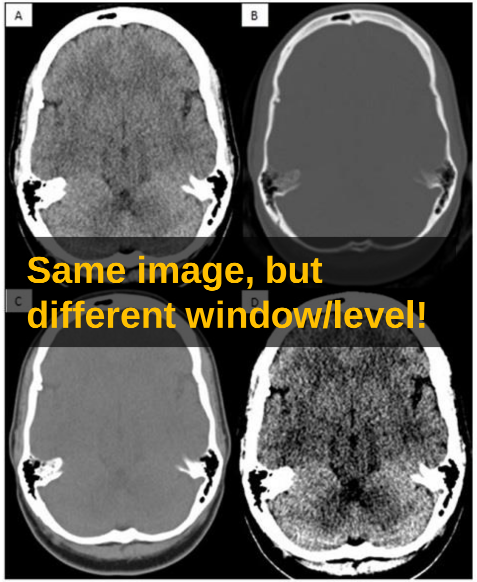
The dynamic range of images

- Natural images (e.g. CIFAR10, ImageNet) are usually 8-bit per channel
  Why? Tradeoff between human perception / storage (8-bit → 255 is nice)

- Medical images **are usually not 8-bit**!
  Depending on protocol, they can be 14,16,32-bit

→ Dynamic range: The span between darkest and brightest value

A Brief **(but Important?)** Aside



The dynamic range of images

- Natural images (e.g. CIFAR10, ImageNet) are usually 8-bit
  Why? Tradeoff between human perception / storage (8-bit -

- Medical images **are usually not 8-bit**!
  Depending on protocol, they can be 14,16,32-bit

→ Dynamic range: The span between darkest and brightest value

→ For dynamic ranges >>8-bit, we usually use window/leveling

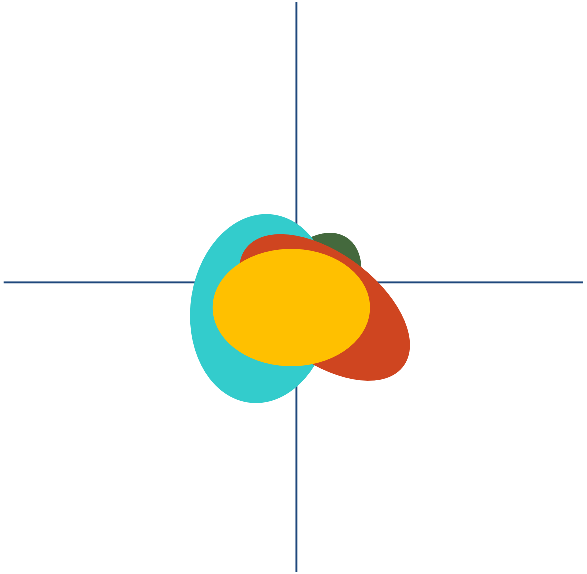
→ When working with such images, do not artificially squash the dynamic range!

Activation, Initialization, Preprocessing, Dropout, Batch Norm

Covariate Shift and Batch Norm

Covariate Shifts

Randomly sampling mini-batches: Training assumes similar distribution!

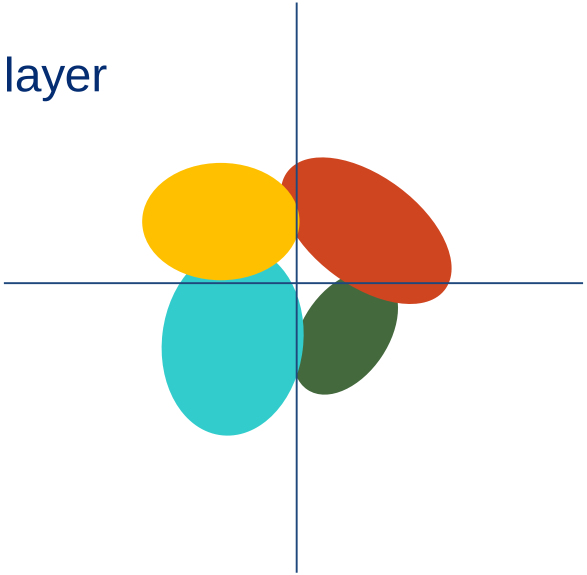


Covariate Shifts

Randomly sampling mini-batches: Training assumes similar distribution!

In practice (and although random), each mini-batch will have different distribution
→ Covariate shift

→ Can happen in **each** layer



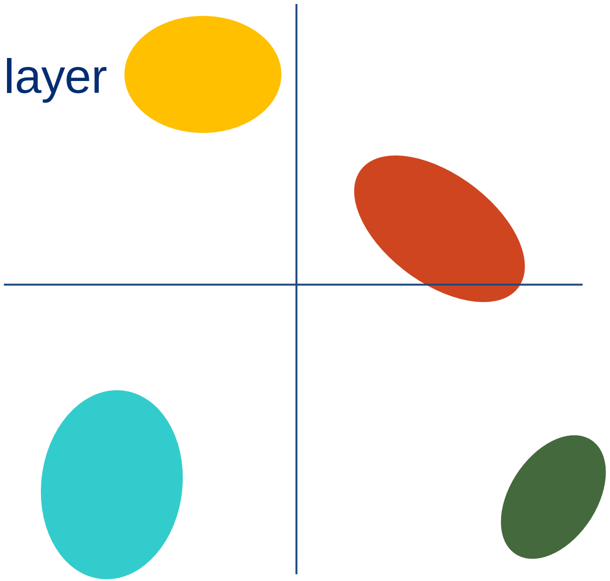
Covariate Shifts

Randomly sampling mini-batches: Training assumes similar distribution!

In practice (and although random), each mini-batch will have different distribution

→ Covariate shift

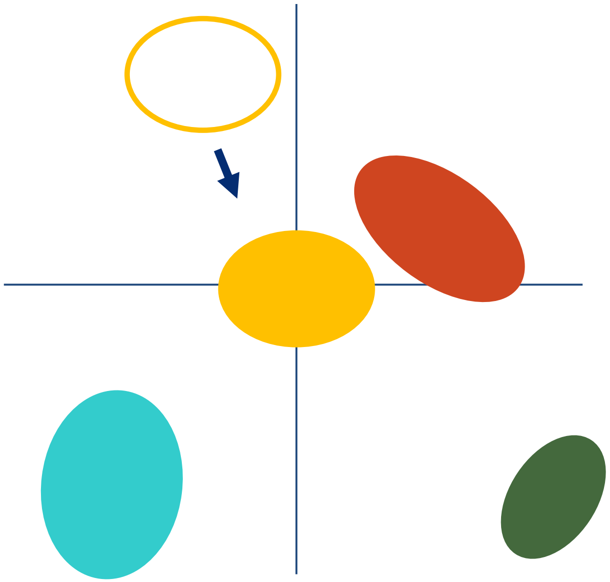
→ Can happen in **each** layer



→ Shifts can be large and can negatively affect training!

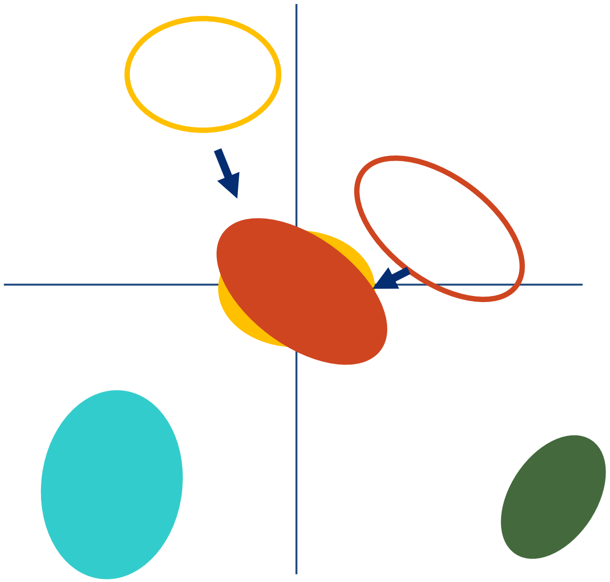
Move Batches to Standard Location

Eliminate covariate shift by “moving” batches to zero mean and unit standard dev



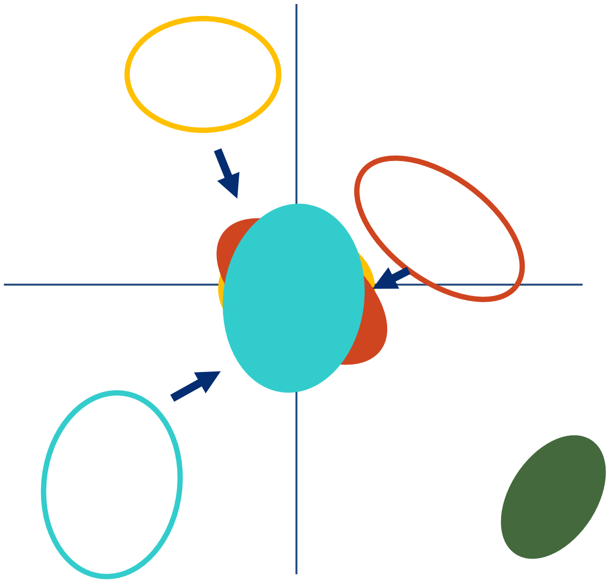
Move Batches to Standard Location

Eliminate covariate shift by “moving” batches to zero mean and unit standard dev



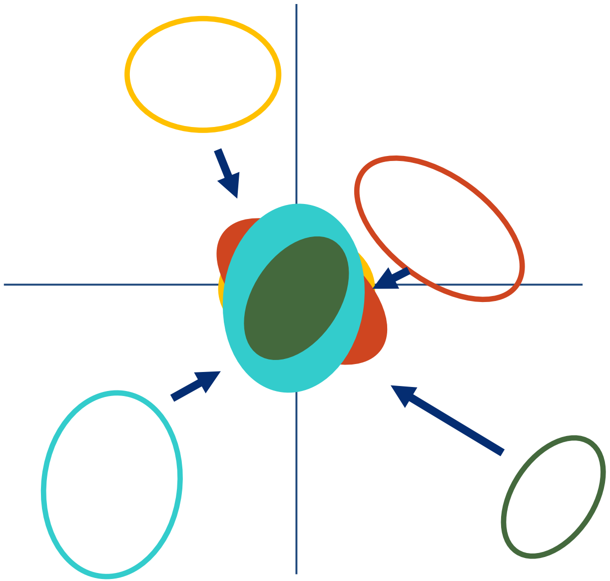
Move Batches to Standard Location

Eliminate covariate shift by “moving” batches to zero mean and unit standard dev



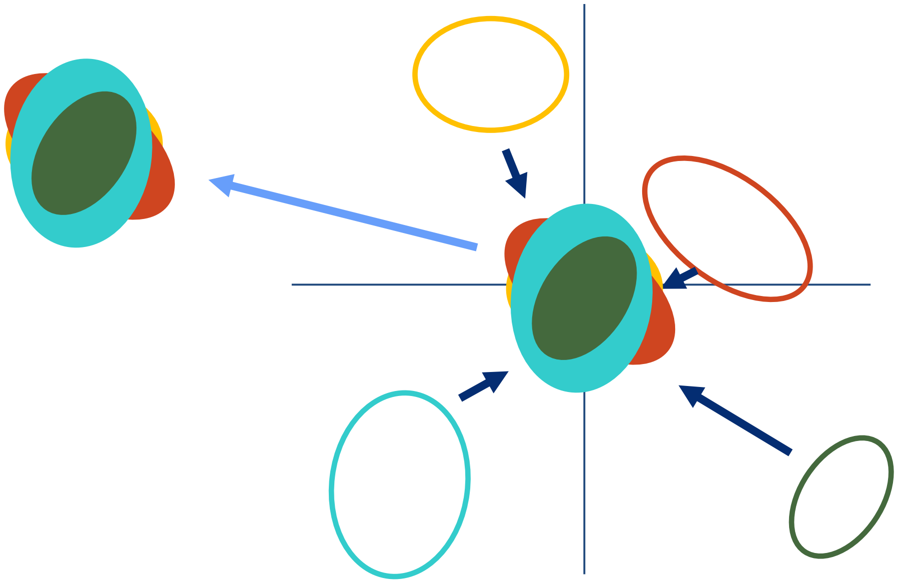
Move Batches to Standard Location

Eliminate covariate shift by “moving” batches to zero mean and unit standard dev



Move Batches to Standard Location

Eliminate covariate shift by “moving” batches to zero mean and unit standard dev



→ Then, move entire collection to desirable location: **Batch normalization**

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167.

Batch Normalization

- If we want unit Gaussian activations, let's make them that!

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

This function is differentiable (backprop!)

- Rather then pre-conditioning data and hoping that nice properties are preserved, at each layer we re-condition during every forward pass

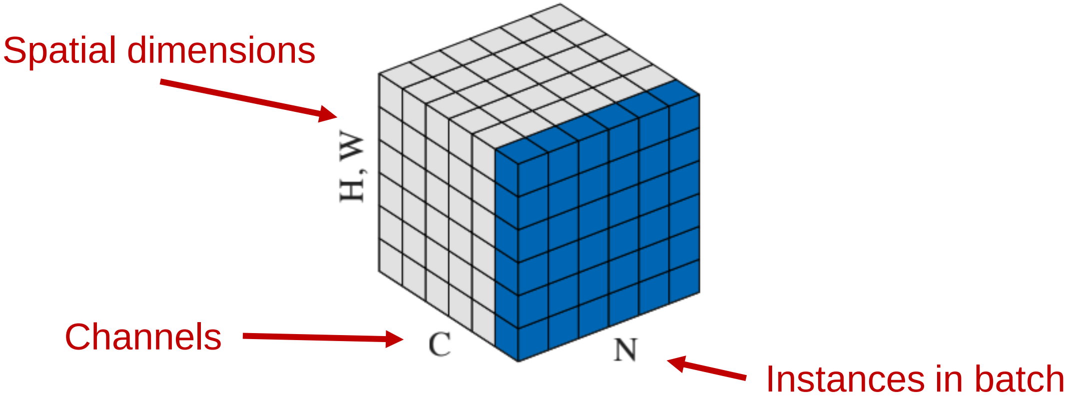
Batch Normalization

- If we want unit Gaussian activations, let's make them that!

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

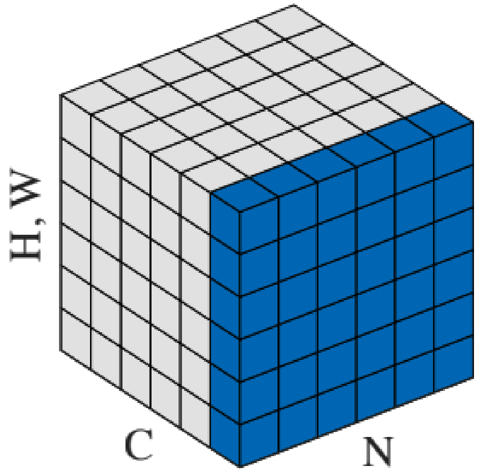
This function is differentiable (backprop!)

- Rather then pre-conditioning data and hoping that nice properties are preserved, at each layer we re-condition during every forward pass



Wu, Y. and He, K., 2018. Group normalization. arXiv preprint arXiv: 1803.08494.

Batch Normalization

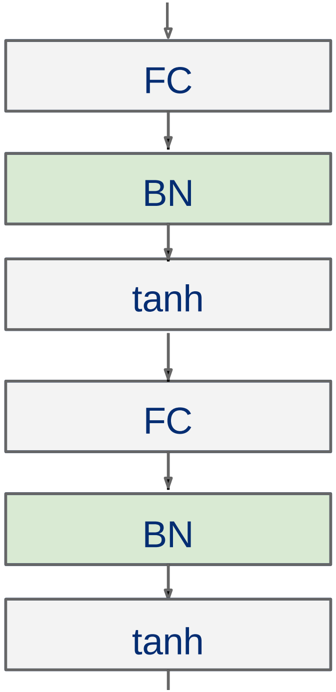


1. Compute empirical mean and variance for each channel
$$E[x^{(k)}], \mathrm{Var}[x^{(k)}]$$

2. Normalize to unit Gaussian
$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Wu, Y. and He, K., 2018. Group normalization. arXiv preprint arXiv: 1803.08494.

Batch Normalization



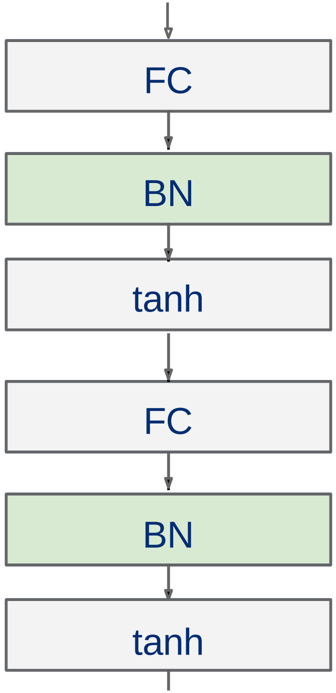
Usually inserted right after fully connected
or convolutional layers, right before activation.

1. Compute empirical mean and variance for each channel
   $$E[x^{(k)}], \mathrm{Var}[x^{(k)}]$$

2. Normalize to unit Gaussian
   $$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Batch Normalization

Usually inserted right after fully connected
or convolutional layers, right before activation.

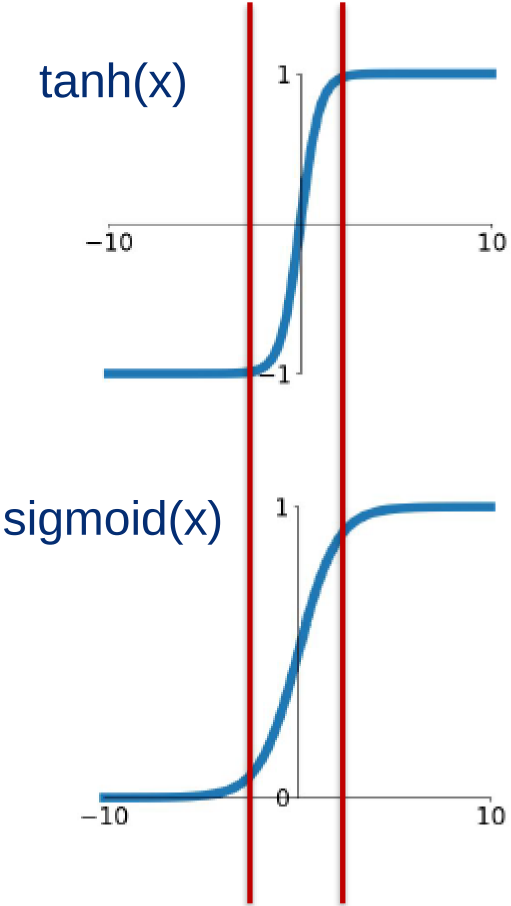


1. Compute empirical mean and variance for each channel
$$E[x^{(k)}], \mathrm{Var}[x^{(k)}]$$

2. Normalize to unit Gaussian
$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Q: Is unit Gaussian activation necessarily what we want?

Batch Normalization



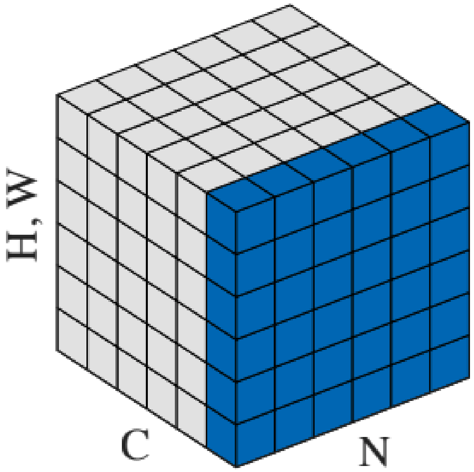
Consider tanh or sigmoid activation

→ Batch normalization will limit the activation to the linear regime of these activation functions!

→ In such case, negatively affects performance

There are other cases where you also would not want BN, e.g. when magnitude matters.

Batch Normalization



1. Compute empirical mean and variance for each channel
$$E[x^{(k)}], \mathrm{Var}[x^{(k)}]$$

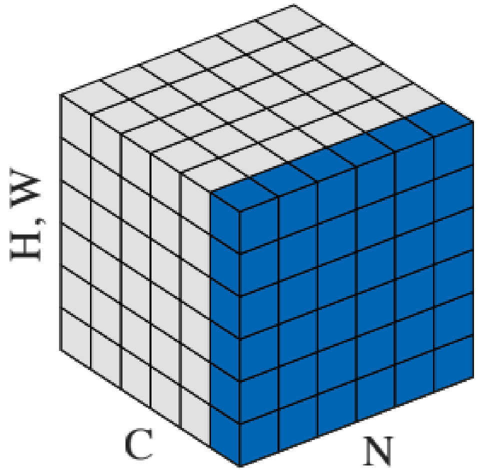
2. Normalize to unit Gaussian
$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

3. Squash output to beneficial range
$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

These are parameters and are learned during training.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167.

Batch Normalization



1. Compute empirical mean and variance for each channel
   $$E[x^{(k)}], \mathrm{Var}[x^{(k)}]$$

2. Normalize to unit Gaussian
   $$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

3. Squash output to beneficial range
   $$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

These are parameters and are learned during training.

Network can learn identity!

$$\gamma^{(k)} = \mathrm{Var}[x^{(k)}]$$
$$\beta^{(k)} = E[x^{(k)}]$$

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167.

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

- Improves gradient flow through network and allows for higher learning rates
  - Avoids saturating activations
  - Avoids exploding/vanishing gradients
  - Higher learning rates usually produce larger weights leading to explosion
    → Can be avoided here since re-normalized
- Reduces strong dependence on initialization
- Acts as regularization
  - Single instance is now seen in conjuncture with other samples of the batch
  - Network outputs per sample no longer deterministic

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167.

Batch Normalization During Testing

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

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

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

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

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

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

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Q: What to do at testing time?

Batch Normalization During Testing

Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$

$N_{BN}^{inf} \leftarrow N_{BN}^{tr}$ // Inference BN network with frozen // parameters

for $k = 1 \dots K$ **do**

// For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.

Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\mathrm{Var}[x] \leftarrow \frac{m}{m-1}\mathrm{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

In N_{BN}^{inf}, replace the transform $y = \mathrm{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$

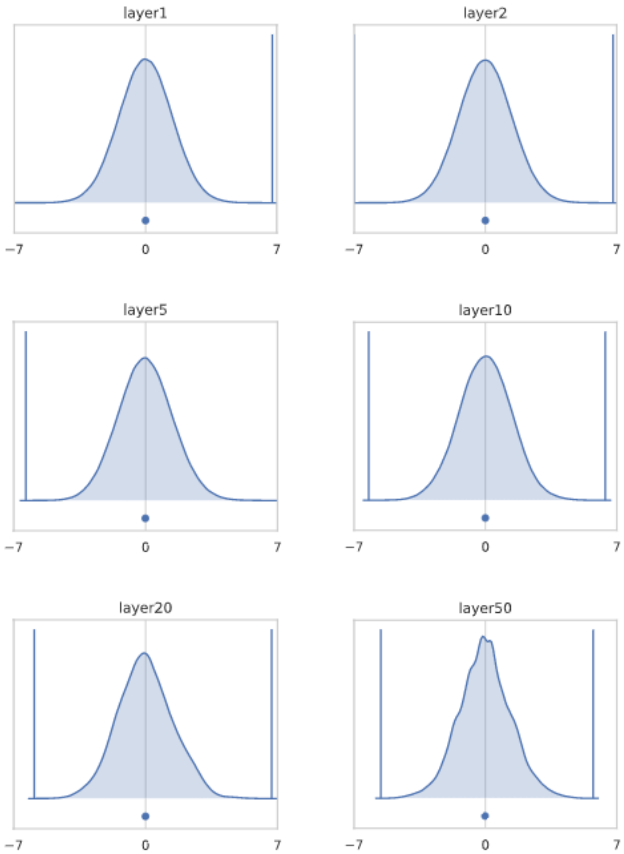
end for

Q: What to do at testing time?

Compute average mean and standard deviation across multiple batches, then save these values for inference.

Batch Norm: New Insights

- Is it really about covariate shift?
- Let's reconsider He initialization
  - Goal: Preserve mean and variance of outputs if marginalized over the weight distribution

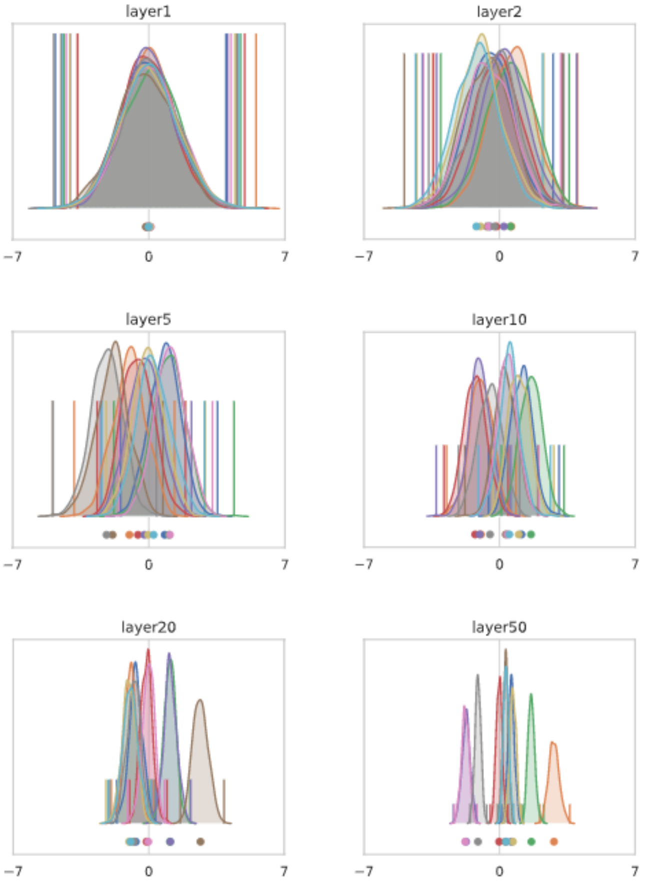


Channel activation at different depths with independent N(0,1) inputs

Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. NeurIPS (pp. 2483-2493).
https://myrtle.ai/how-to-train-your-resnet-7-batch-norm/

Batch Norm: New Insights

- Is it really about covariate shift?
- Let's reconsider He initialization
  - Goal: Preserve mean and variance of outputs if marginalized over the weight distribution
- Every channel has chosen a constant value!
  - Peaked, narrow distribution
  - Most inputs would be classified as the orange class

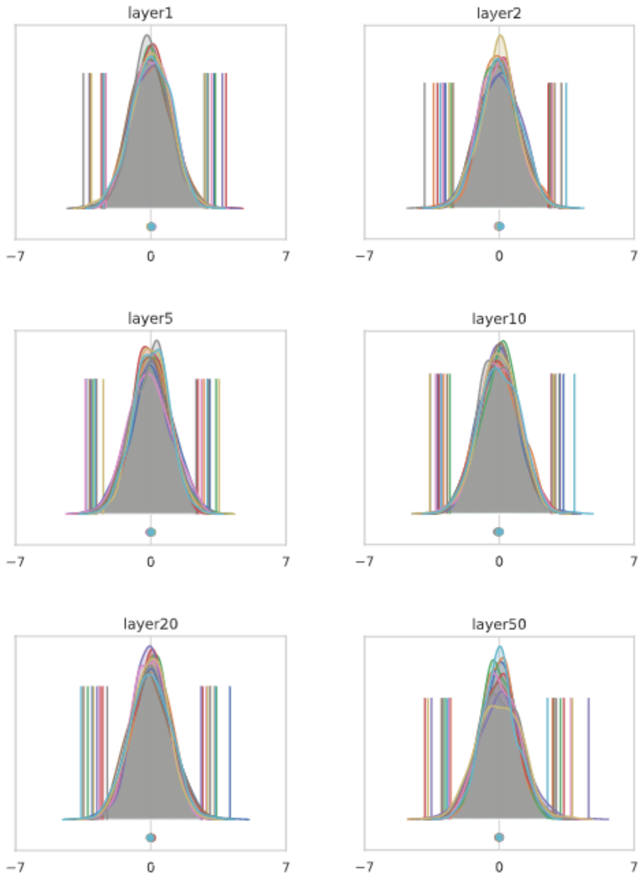


Channel activation at different depths with independent N(0,1) inputs **split by channel**

Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. NeurIPS (pp. 2483-2493).
https://myrtle.ai/how-to-train-your-resnet-7-batch-norm/

Batch Norm: New Insights

- Is it really about covariate shift?
- Let's reconsider He initialization
  - Goal: Preserve mean and variance of outputs if marginalized over the weight distribution
- Every channel has chosen a constant value!
  - Peaked, narrow distribution
  - Most inputs would be classified as the orange class
- Removing ReLU: Problem disappears
  - Non-zero channel means
  - Decreasing variance due to increasing mean (see blog)



Channel activation at different depths with independent N(0,1) inputs

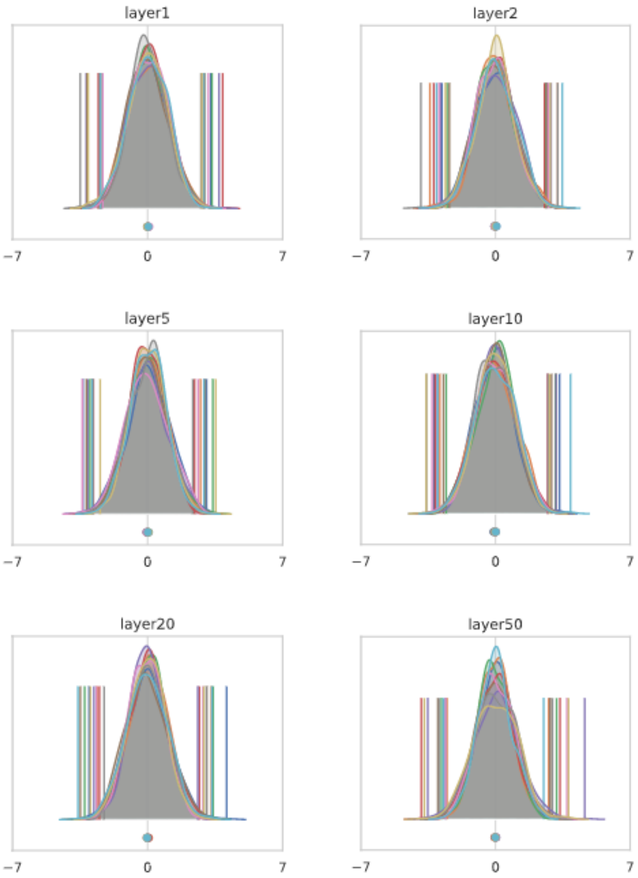
split by channel, no ReLU

Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. NeurIPS (pp. 2483-2493).
https://myrtle.ai/how-to-train-your-resnet-7-batch-norm/

Batch Norm: New Insights

- → Without batch norm
  - → Standard initialization leads to bad configurations
  - → Network will predict near constant outputs
- → Batch norm fixes this by design

What happens during training?



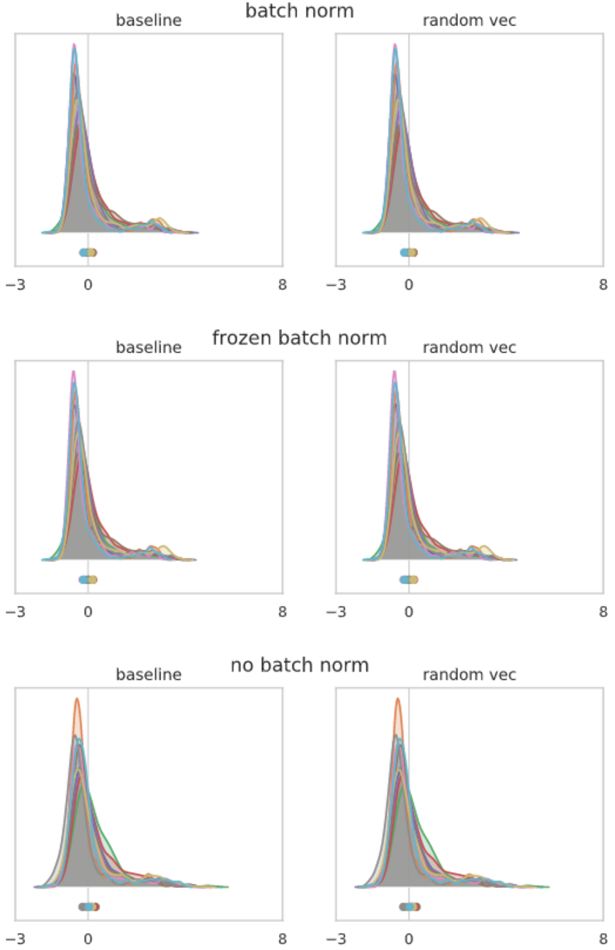
Channel activation at different depths with independent N(0,1) inputs

split by channel, no ReLU

[Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. NeurIPS (pp. 2483-2493).](#)
https://myrtle.ai/how-to-train-your-resnet-7-batch-norm/

Batch Norm: New Insights

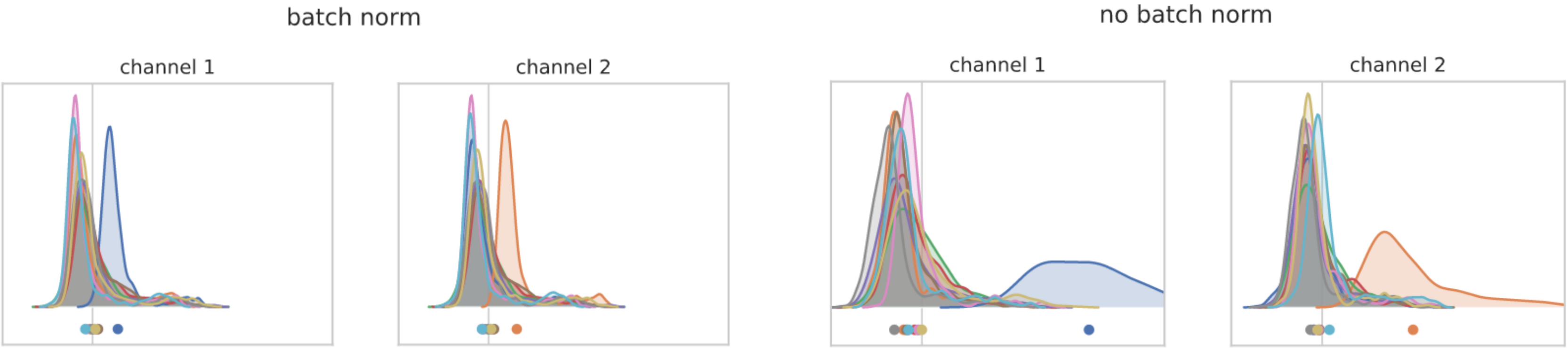
- Random perturbation of the weight
  Strength of 1% of parameter vector length
  - Similar output distributions
  - Main mode and second smaller mode: Network starting to make confident predictions



Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. NeurIPS (pp. 2483-2493).
https://myrtle.ai/how-to-train-your-resnet-7-batch-norm/

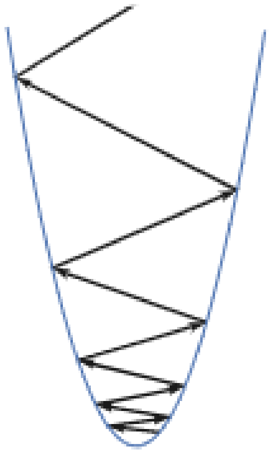
Batch Norm: New Insights

- **Targeted** perturbation of the weight
  Strength of 1% of parameter vector length
  Gradient of channel mean
  - Network will predict perturbed class in majority of inputs!
  - Internal covariate shift can propagate to external covariate shift in non-batch norm networks!



Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. NeurIPS (pp. 2483-2493).
https://myrtle.ai/how-to-train-your-resnet-7-batch-norm/

Batch Norm: New Insights



- **Targeted** perturbation of the weight
  Strength of 1% of parameter vector length
  Gradient of channel mean
  - Network will predict perturbed class in majority of inputs!
  - Internal covariate shift can propagate to external covariate shift in non-batch norm networks!

- What does this mean for optimization?
  - Without batch norm, small perturbations lead to immense increases in loss!
  - This means that we are in a narrow valley-type loss landscape (see also next lecture)

[Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. NeurIPS (pp. 2483-2493).](https://myrtle.ai/how-to-train-your-resnet-7-batch-norm/)
https://myrtle.ai/how-to-train-your-resnet-7-batch-norm/

Batch Norm: New Insights

- Investigate the Hessian of parameters
  - **Leading eigenvector** (direction of largest curvature)
    → This direction makes SGD spiral out of control
  - Computed via a power method (not important)
- Also, compute gradients w.r.t.
  mean channel activation (as in perturbation)

→ **Compute overlap between eigenvectors and output-mean gradients**

Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. NeurIPS (pp. 2483-2493).
https://myrtle.ai/how-to-train-your-resnet-7-batch-norm/

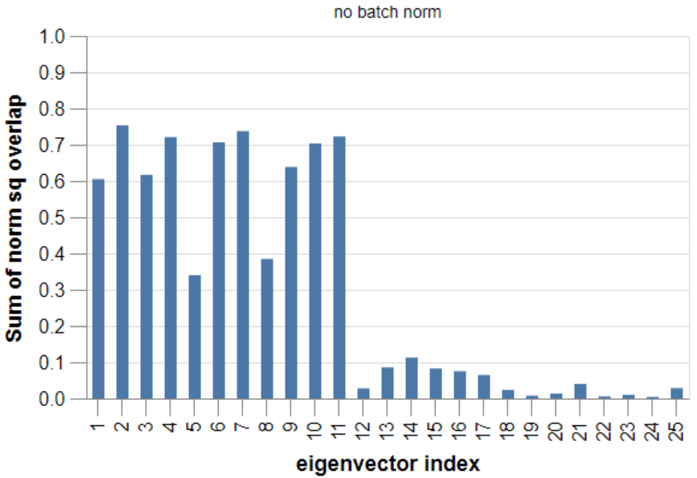
Batch Norm: New Insights

→ **Compute overlap between eigenvectors and output-mean gradients**

- Largest eigenvectors lie almost entirely in the 9-dim subspace spanned by the mean-output gradients!

- **This de-stabilizes SGD optimization!**

- **Batch norm: Smoothens the optimization landscape.**



Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. NeurIPS (pp. 2483-2493).
https://myrtle.ai/how-to-train-your-resnet-7-batch-norm/

Activation, Initialization, Preprocessing, Dropout, Batch Norm

Regularization with Dropout

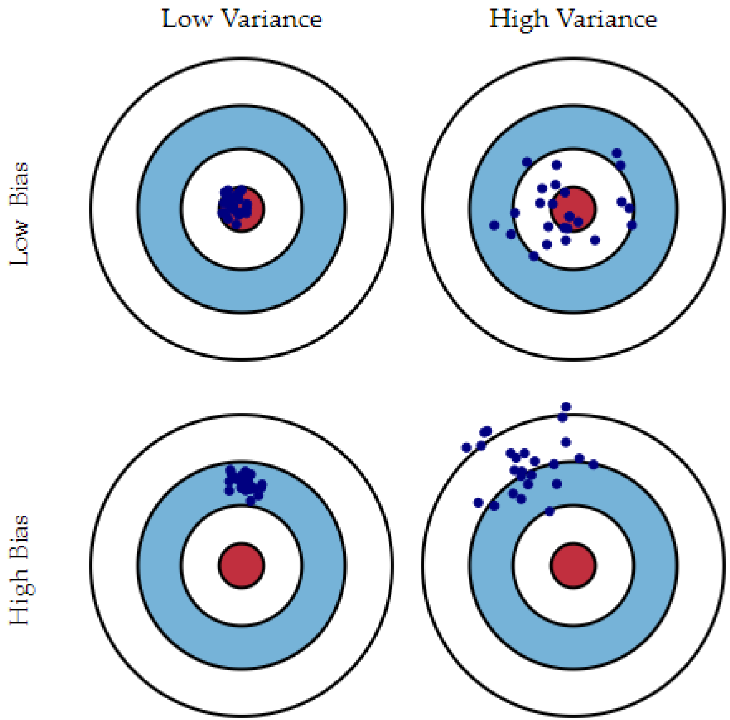
The Bias-Variance Tradeoff and Regularization

Decomposition into bias and variance

$$L(W) = \underbrace{(E[\hat{y}] - y)^2}_{\text{Bias}^2} + \underbrace{E[(\hat{y} - E[\hat{y}])^2]}_{\text{Variance}} + \sigma$$

Adding regularization

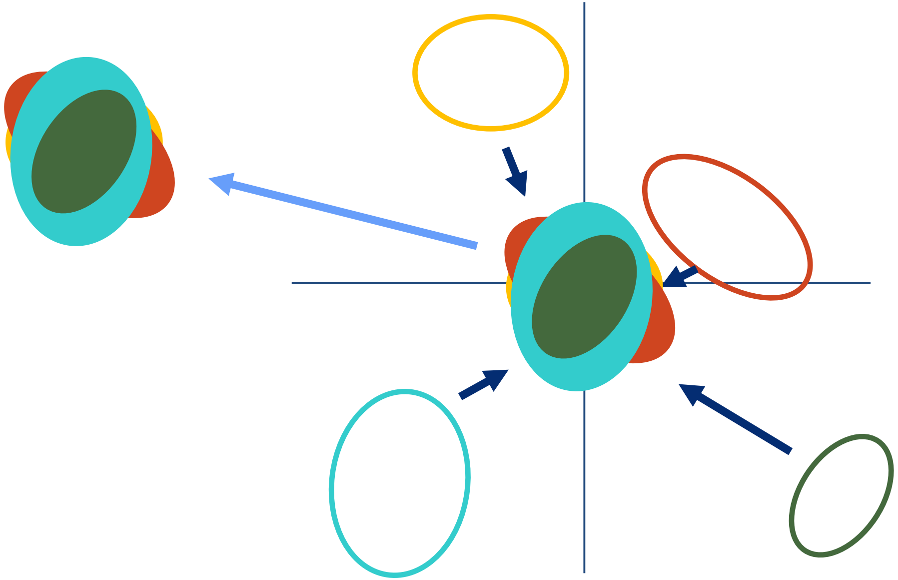
$$L(W) = \underbrace{\frac{1}{N}\sum_i L_i\left(f(x_i, W), y_i\right)}_{\textbf{Data fidelity}} + \underbrace{\lambda R(W)}_{\textbf{Regularization}}$$



[Hastie, T., Tibshirani, R, and Friedman, J. (2017) The Elements of Statistical Learning]

Batch Normalization

Regularization “in a funny way” by seeing samples in conjuncture with others



Other approaches

- L2 on weights
- L1 on activations
- Adding noise to inputs

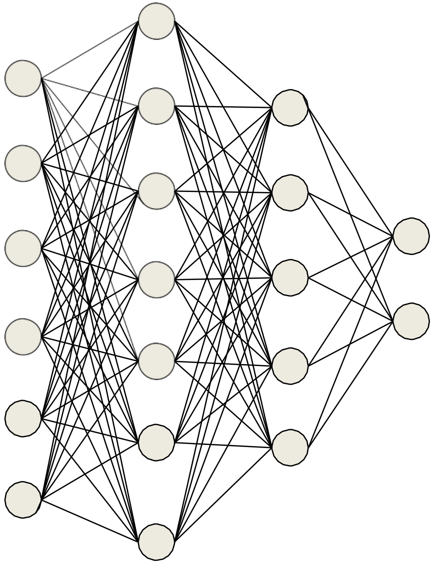
Q: Can we regularize “in a less funny” way?

[Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167.]

Dropout

No, not like this!

Dropout

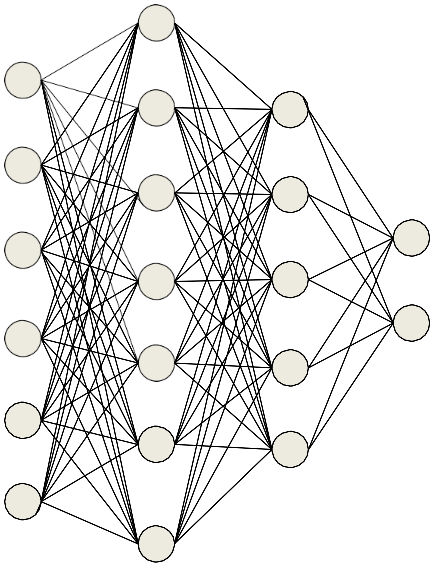


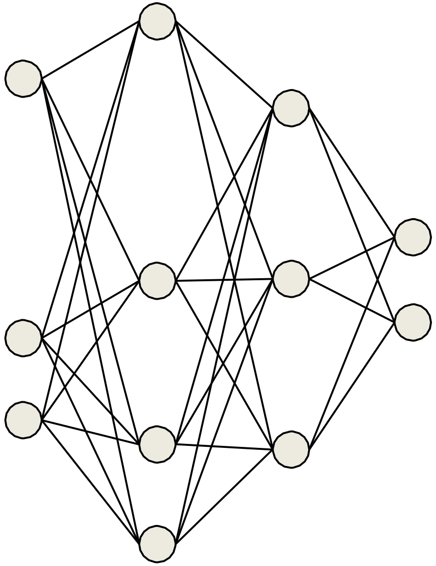
During training

- At each iteration, in each layer, “knock out” each neuron with probability 1-α

[Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. JMLR, 15(1), 1929-1958.]

Dropout



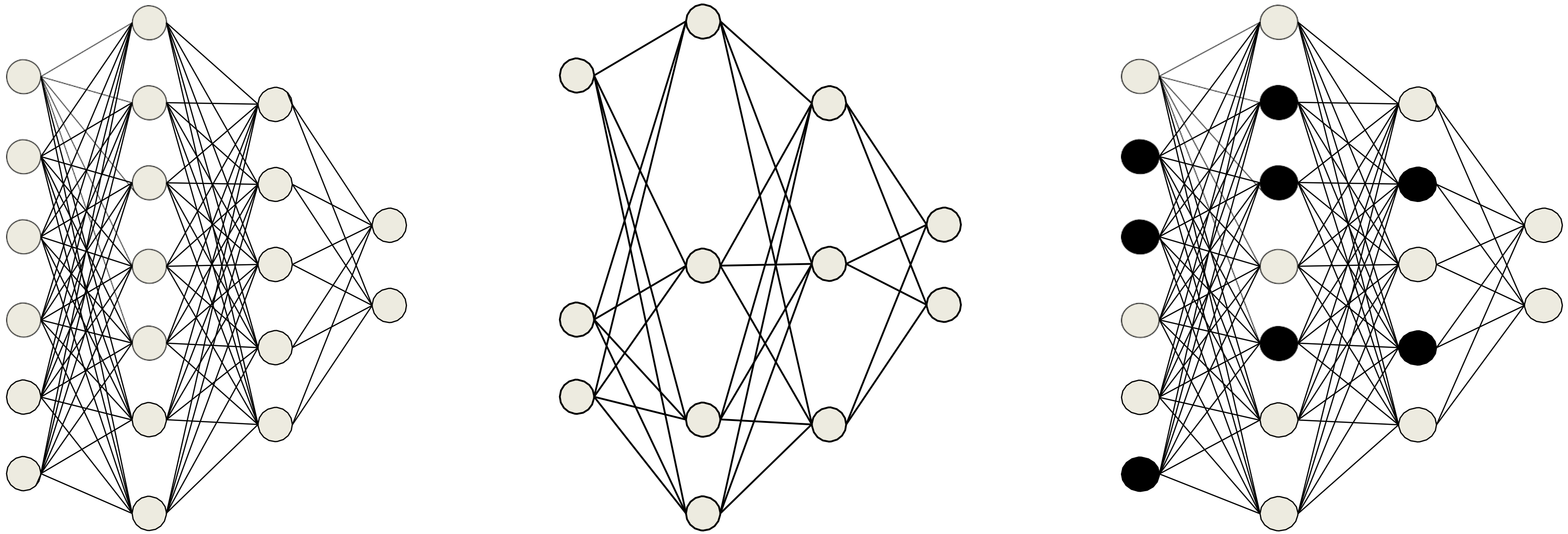


During training

- At each iteration, in each layer, “knock out” each neuron with probability 1-α

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. JMLR, 15(1), 1929-1958.

Dropout



During training

- At each iteration, in each layer, “knock out” each neuron with probability 1-α
- In practice, we do not drop connections but set inputs/outputs to zero

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. JMLR, 15(1), 1929-1958.

Dropout in Forward Pass

Without dropout:

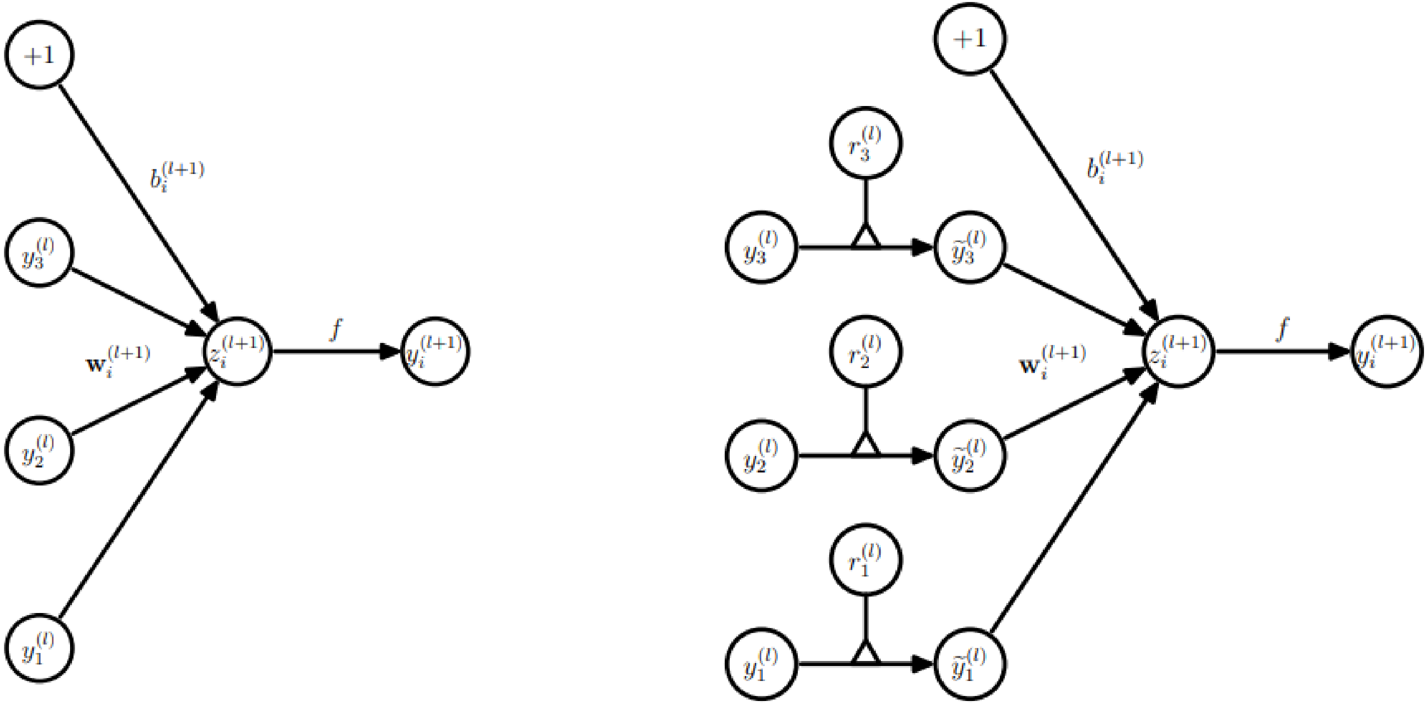
$$
\begin{aligned}
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \mathbf{y}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}),
\end{aligned}
$$

With dropout:

$$
\begin{aligned}
r_j^{(l)} &\sim \mathrm{Bernoulli}(p), \\
\widetilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \widetilde{\mathbf{y}}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}).
\end{aligned}
$$

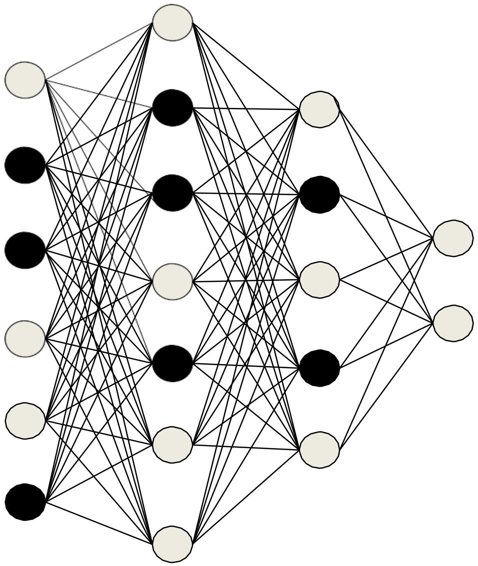
1. For every node j and layer l, determine Bernoulli number {0,1}
2. Drop outputs
3. ???
4. Profit.

Dropout in Forward Pass



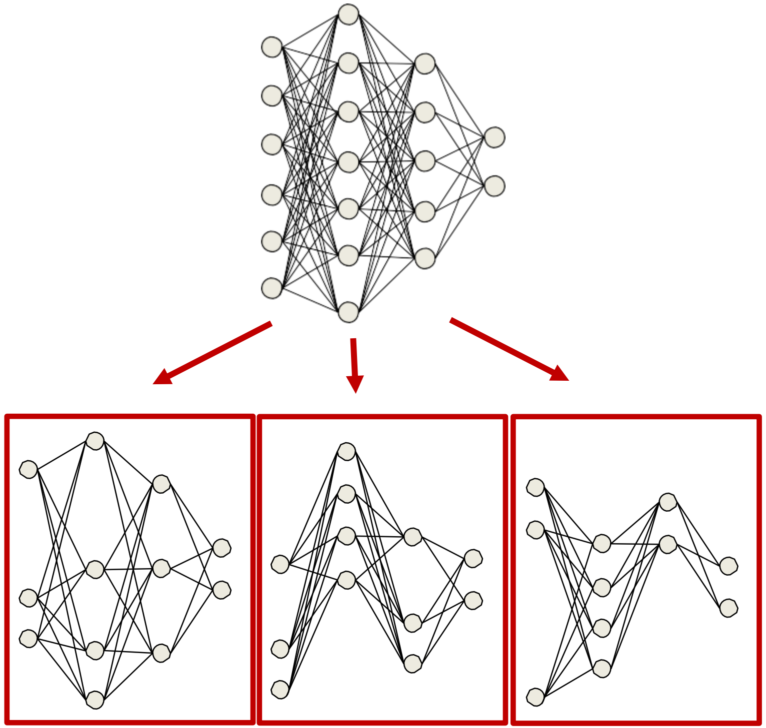
(a) Standard network (b) Dropout network

Dropout in Backward Pass



- Backpropagation as usual, but
  Set updates to zero for dropped out weights
- Tricks of the trade still work

Dropout at Inference Time

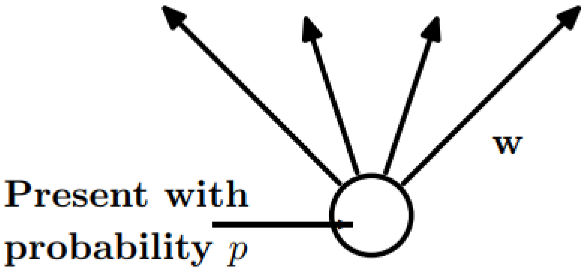


A slightly different view onto dropout

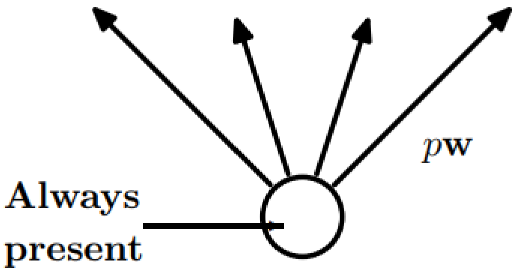
- 2^N sub-networks for N-neuron network
- Dropout samples over these sub-networks

→ Learns a network that averages over all possible networks

Dropout at Inference Time



(a) At training time



(b) At test time

During training

- Fewer activations present

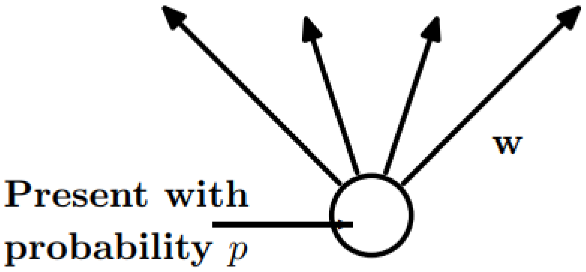
→ Overall activation smaller

During testing

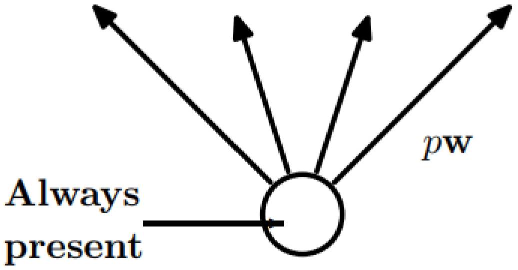
- All activations present

→ Weights or activations scaled by *p*

Dropout at Inference Time



(a) At training time



(b) At test time

During training

- Fewer activations present

→ Overall activation smaller

During testing

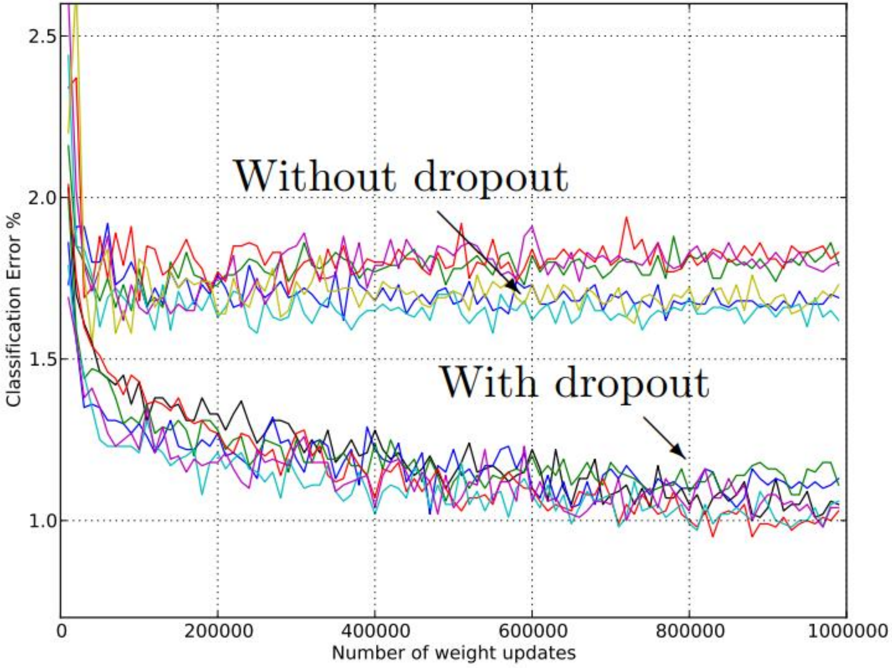
- All activations present

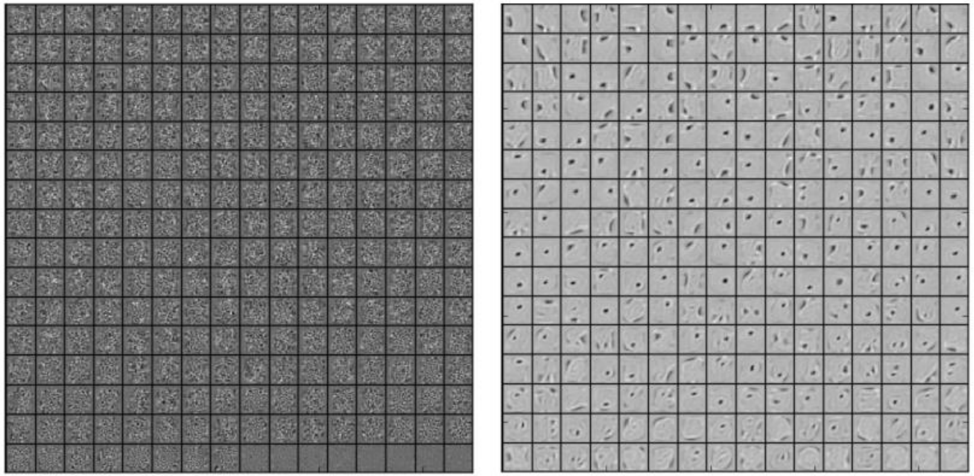
→ Weights or activations scaled by *p*

Some research on test time dropout.

Q: Why would you want to do this?

Dropout: Typical Values and Results





(a) Without dropout (b) Dropout with $p = 0.5$.

This experiment considers an autoencoder. We will see this behavior again later.

Typical values

- Input unit dropout: 0.2
- Hidden unit dropout: 0.5

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. JMLR, 15(1), 1929-1958.

Activation, Initialization, Preprocessing, Dropout, Batch Norm

Questions?