COMP/EECE 7/8740 Neural Networks

Topics: Efficient Training Approaches

- Input pre-processing (augmentation)
- Initialization approaches
- Advanced activation functions
- Regularization and
- Batch normalization

Md Zahangir Alom
Department of Computer Science
University of Memphis, TN

Overview : Learning with DNN

- Pre-processing

- **Initialization**
- **Advanceda activation function**
- **Regularization**
- **Batch normalization**
- Optimization techniques
- Loss functions

- Alternative FC
- Average pooling
- Global Average pooling

Inputs

Related to the DNN models implementation and ensuring efficient training

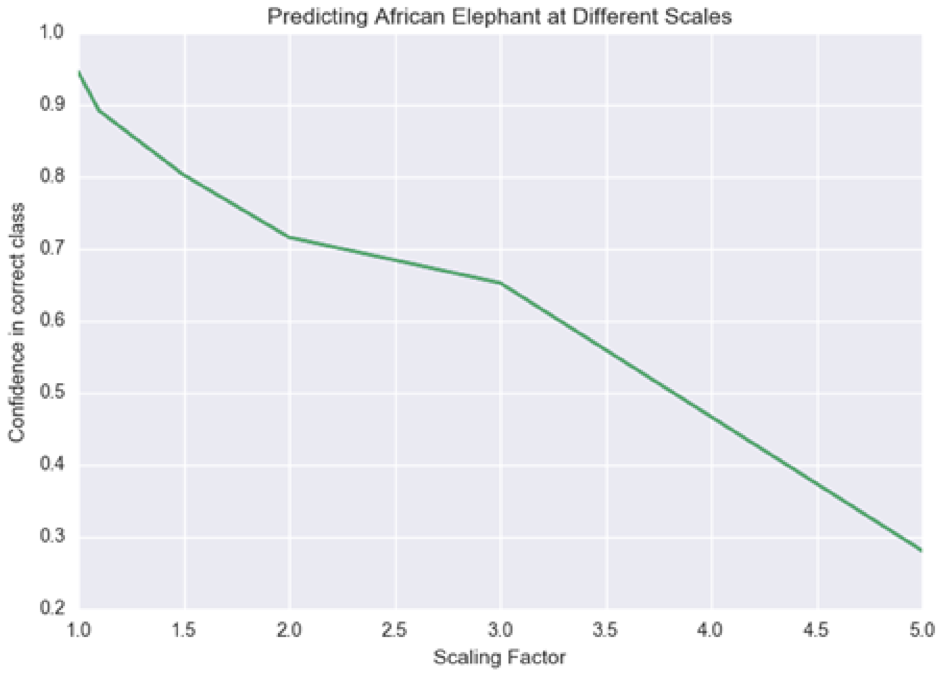
Related to classification & output layers

Impact of input size

- Input this picture of an elephant to ResNet-50 at various scales
- ResNet50 was trained on 224x224 images
- Question?
  - How much bigger can I make the image before the elephant is misclassified?

Impact of input size

- I tried **rescales of** [1.1, 1.5,3,5,10]
- Elephant was **correctly classified up till 5x** scaling
  - Input size was 1120x1120
- **Confidence of classification decays slowly**, at rescale factor of 10, ‘African Elephant’ is **no longer in the top 3**



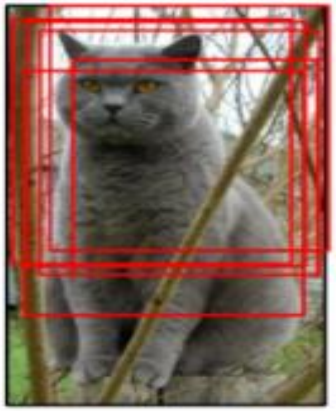
Data augmentation

Image **data augmentation** is a technique that can be used to **artificially expand the number** of a training dataset by **creating modified versions of images** in the dataset

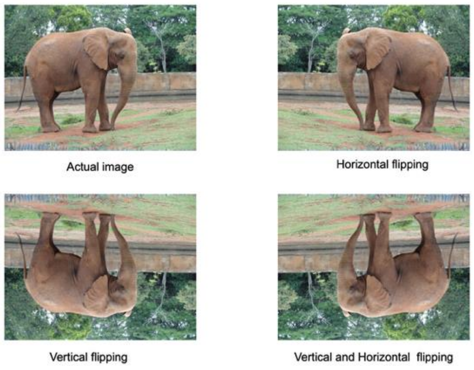
Preprocessing methods

- Sample Normalization
- Mean subtraction
- Random **cropping/ rescaling**
- Flipping sample with respective to the horizon or vertical axis
- Color jittering
- PCA/ZCA whitening
- many more. …

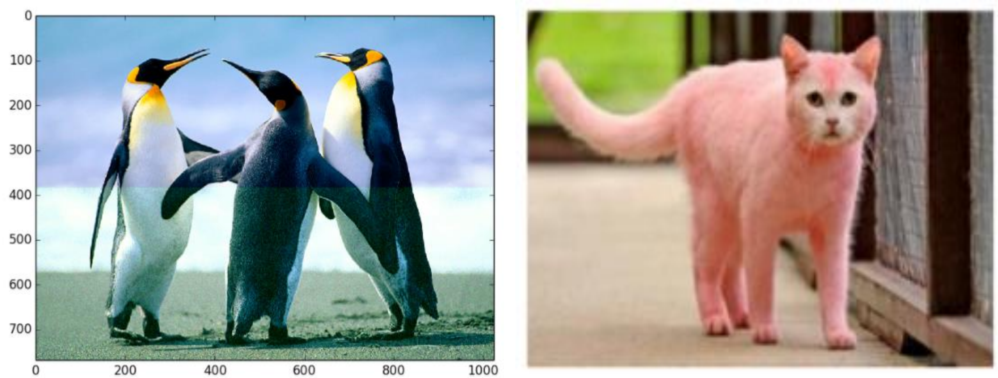
- Max_norm : $I_{mxn} = \frac{I}{V_{max}}$
- Max_Min_norm : $I_{mxn} = \frac{I \; - \; V_{min}}{V_{max} - V_{min}}$
- Rescaled : $I_R = \frac{I}{V_{max}} * R_f$



Random cropping



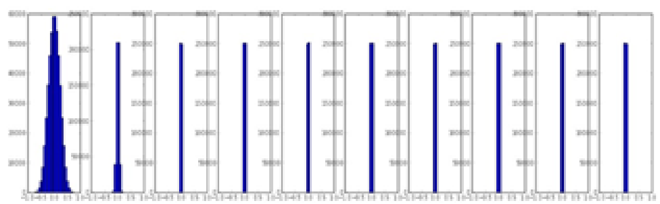
Flipping sample



Color jittering

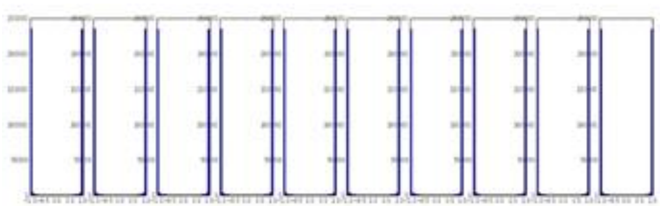
Weight Initialization

- **Basic initialization**
- Smarter initialization schemes
- Pretraining and
- Transfer Learning



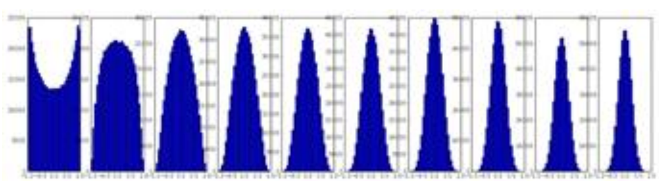
Initialization too small:

- Activations go to zero
- gradients also zero
- No learning



Initialization too big:

- Activations saturate (for tanh)
- Gradients zero,
- no learning



Initialization just right:

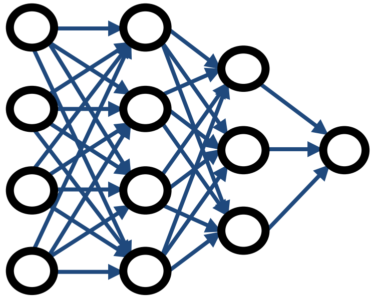
- Nice distribution of activations at all layers,
- Learning proceeds nicely

Baseline and smart Initialization

- **Baseline initialization approaches**: Weights cannot be initialized to same value because the **gradients will be the same**
  - Instead, **draw from some distribution**
    - Uniform from [-0.1, 0.1] is a reasonable starting spot
  - Biases may need special constant initialization

- **Smart initialization approaches**: Weights should be randomly drawn a distribution (e.g. uniform) with mean zero and standard deviation :

$$\sigma = \frac{1}{\sqrt{\text{fan-in}}}$$



fan_in the number of collections feeding into the node.

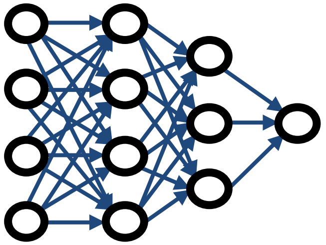
LeCun, Y., Bottou, L., Orr, G. B., and Muller, K. (1998a). Efficient backprop. In Neural Networks, Tricks of the Trade.

What’s Xavier initialization?

- Xavier initialization **makes sure the weights are ‘just right’**, keeping the signal in **a reasonable range of values through many layers.**

- Initialize the weights in a network will be drawn from **a distribution with zero mean and a specific variance**,

$$var(W) = \frac{1}{n_{in}}$$



where W is the initialization distribution for the neuron in question and $\boldsymbol{n_{in}}$ **is the number of neurons feeding into it**.

The distribution used is typically Gaussian or uniform.

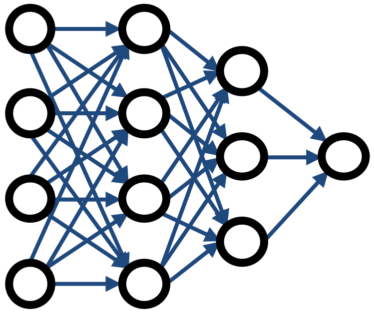
Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." International conference on artificial intelligence and statistics. 2010.

Glorot & Bengio's initialization scheme

- Proper initialization of deep networks is important because of the **multiplicative effect through layers**
- This technique satisfy objectives of maintaining **activation variances and back-propagated gradients variance** as one moves up or down the network.

- We call it the **normalized initialization**:

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]$$



Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." International conference on artificial intelligence and statistics. 2010.

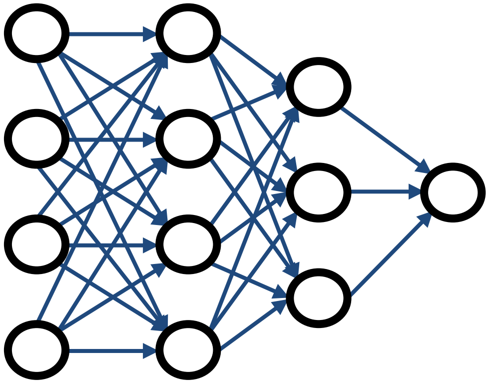
Glorot & Bengio's initialization scheme

- For **TahH** units: sample a Uniform (-r, r) with

$$r = \sqrt{\frac{6}{N_{in} + N_{out}}}$$

- For **sigmoid** units: sample a Uniform (-r, r) with

$$r = 4\sqrt{\frac{6}{N_{in} + N_{out}}}$$

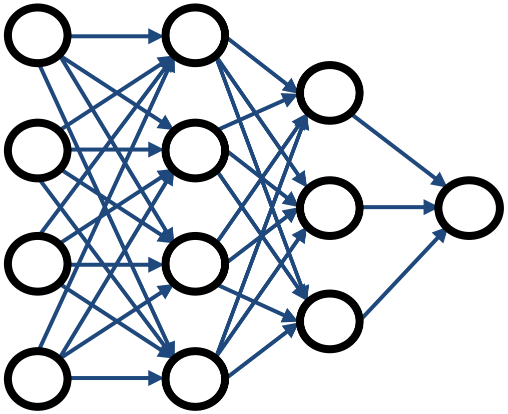


Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 2010.

He initialization (best one)

$$w_l \sim \mathcal{N}\left(0, \frac{2}{\boxed{n_l}}\right)$$

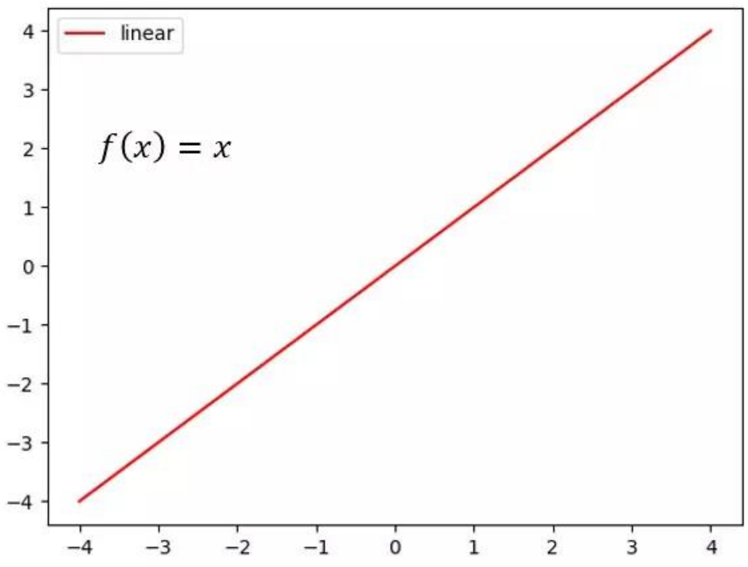
Number of inputs to neuron



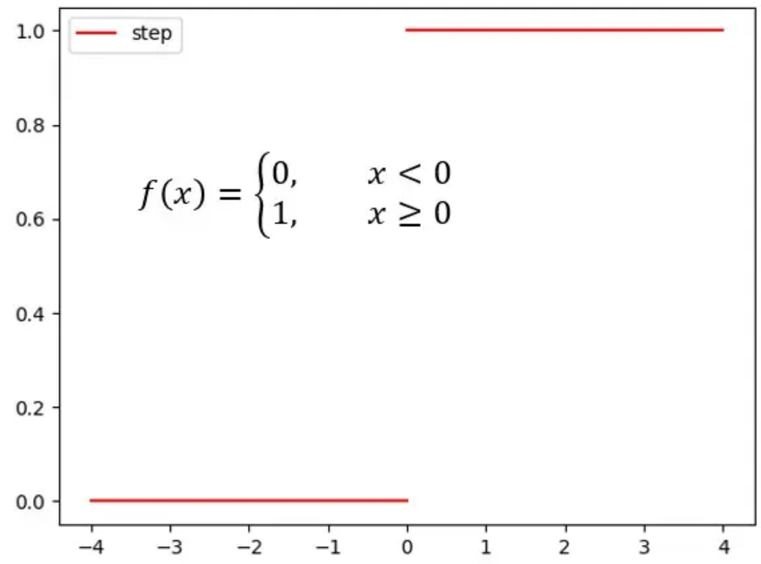
He et al. 2015. "Delving deep into rectifiers: surpassing human level performance on ImageNet classification"

Activation Functions

$$\vec{y} = \text{NeuralNetwork}(\vec{x}) = \sigma(\sigma(\sigma(\sigma(\vec{x} \cdot W_1) \cdot W_2) \cdot W_3) \cdot W_4)$$



Linear activation

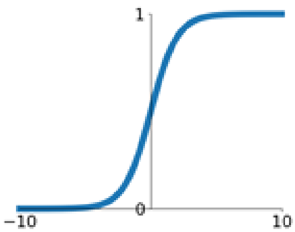


Step function

Activation Functions

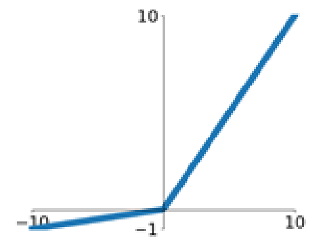
Sigmoid

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



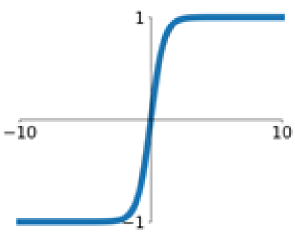
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

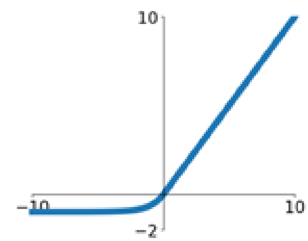


Maxout

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

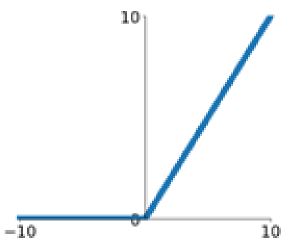
ELU

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



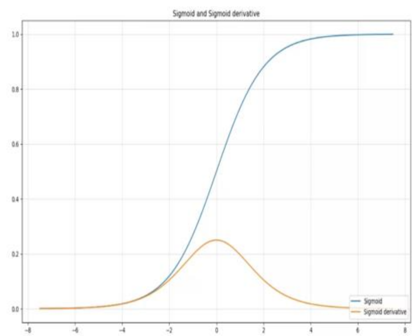
ReLU

$$\max(0, x)$$

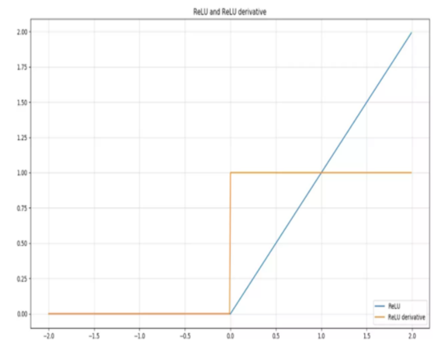


Derivative of activation functions

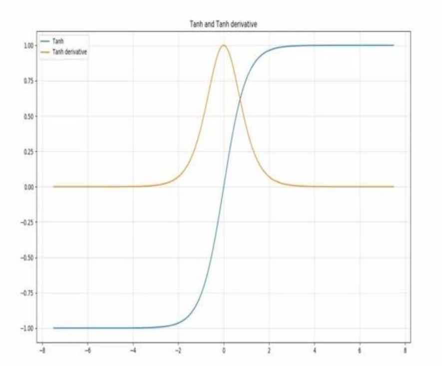
- The derivative of the **ReLU function is one at every point above zero**, creating a more stable network.



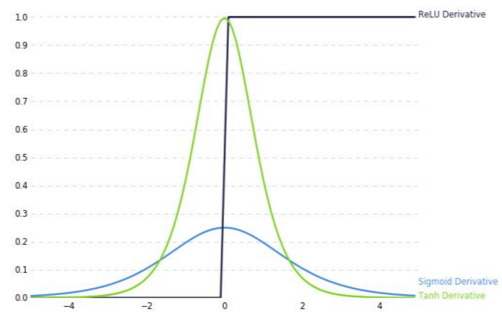
sigmoid



ReLU



tanh



comparison

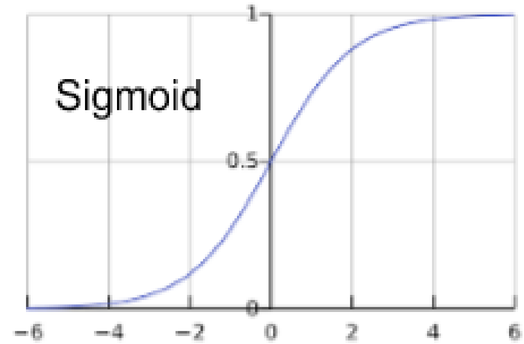
Issues : exploding and vanishing Gradients for Tanh and sigmoid activations !

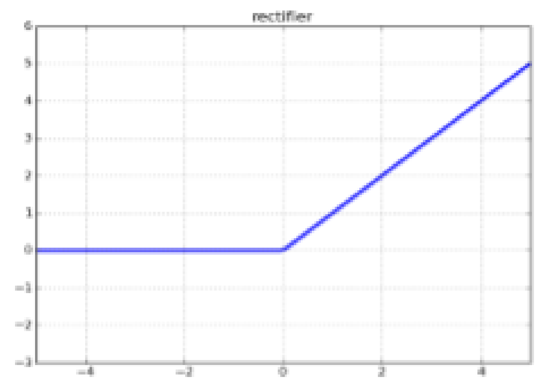
Problem with saturation

Issue:

- A null gradient results in no learning, which happens if:

  1. Sigmoid saturates or
  2. ReLU saturates





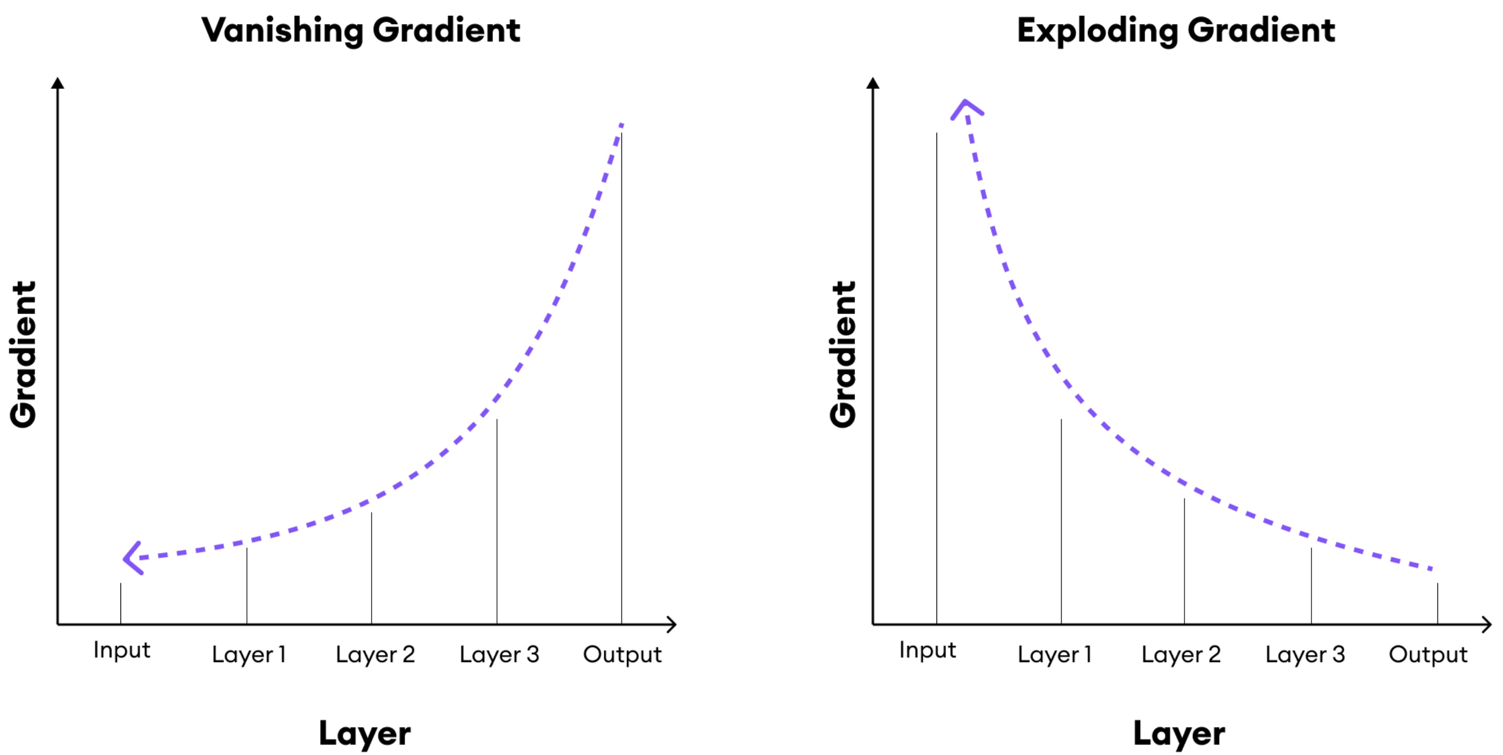
$$RELU(x) = \begin{cases} 0 \text{ if } x < 0 \\ x \text{ if } x >= 0 \end{cases}$$

Solution:

- Initialize the weights so that the average value is zero, i.e., work in the interesting zone of activation function
- Normalize data (zero mean)

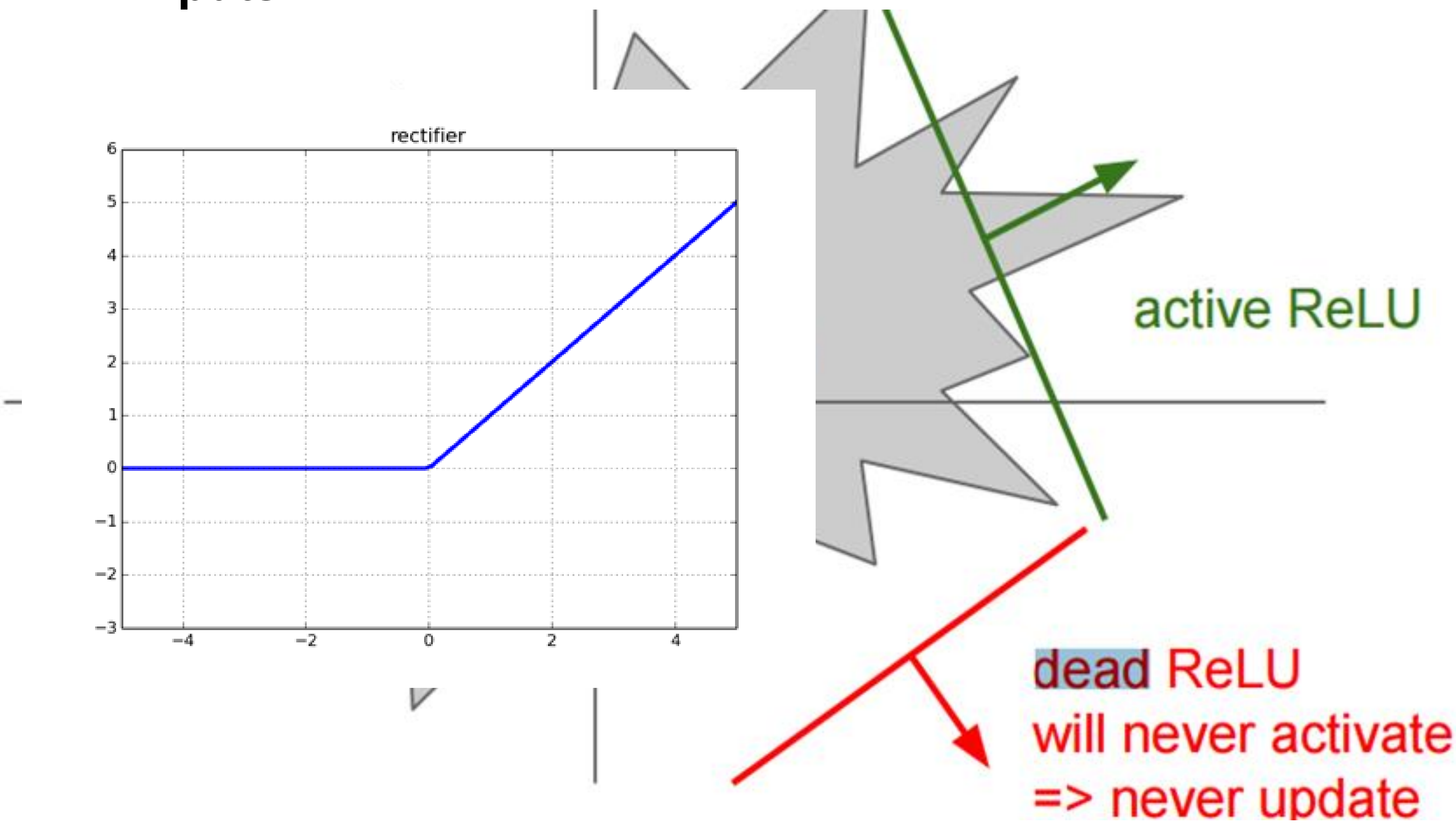
Gradient exploding and saturation

- Gradient status within the networks (respect to the input and output layers)



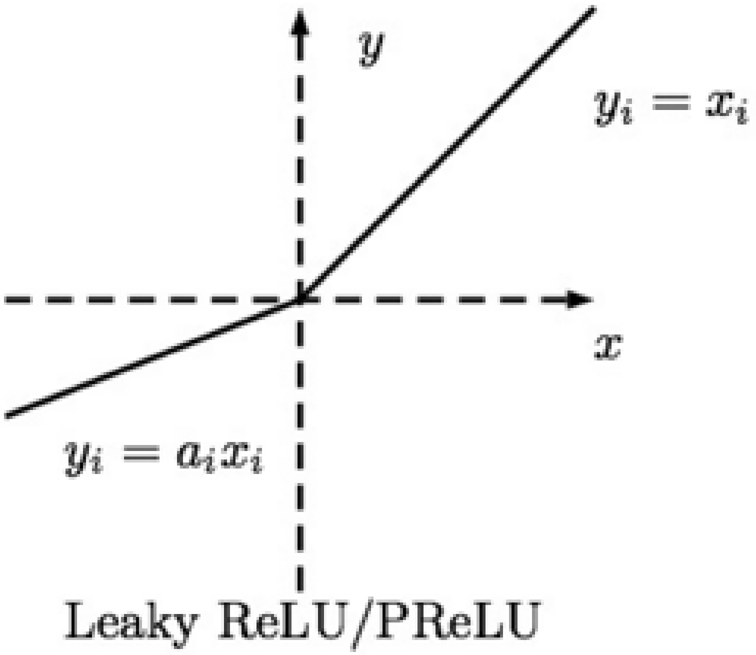
Dying ReLU Problem

- **ReLU dies**, if input to ReLU is negative for the dataset
- Brief burst of **research into addressing dying ReLUs**
- General idea is to have **non-zero gradients even for negative inputs**



Leaky ReLU & Parameterized ReLU

- In Leaky ReLU, **a** is a hyperparameter.
- In Parameterized ReLU(PReLU), **a** is learned.

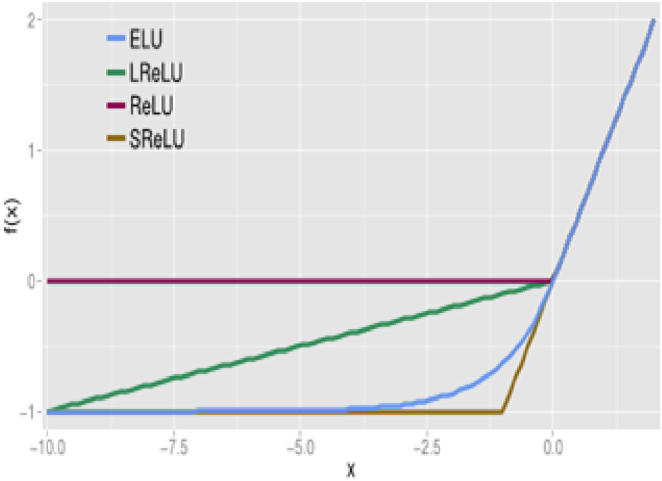


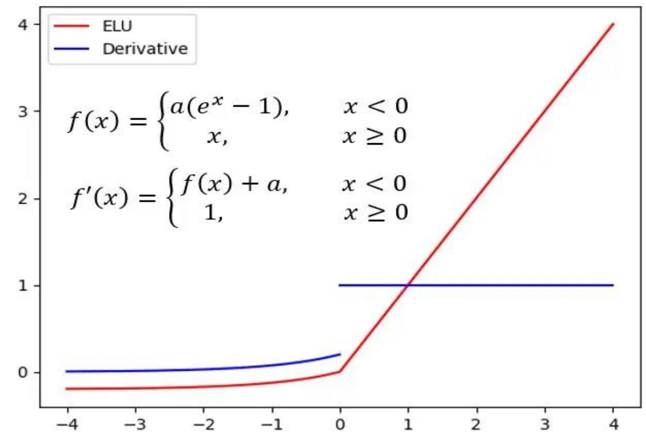
Djork-Arné Clevert, Thomas Unterthiner, Sepp Hochreiter Fast and Accurate Deep Network Learning by Exponential Linear Units

ELU and other activation functions

- The papers introducing each **alternative activations claim they work well**
- All the architectures, we are about to discuss used S**ReLUs**

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}, \quad f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ f(x) + \alpha & \text{if } x \le 0 \end{cases}$$



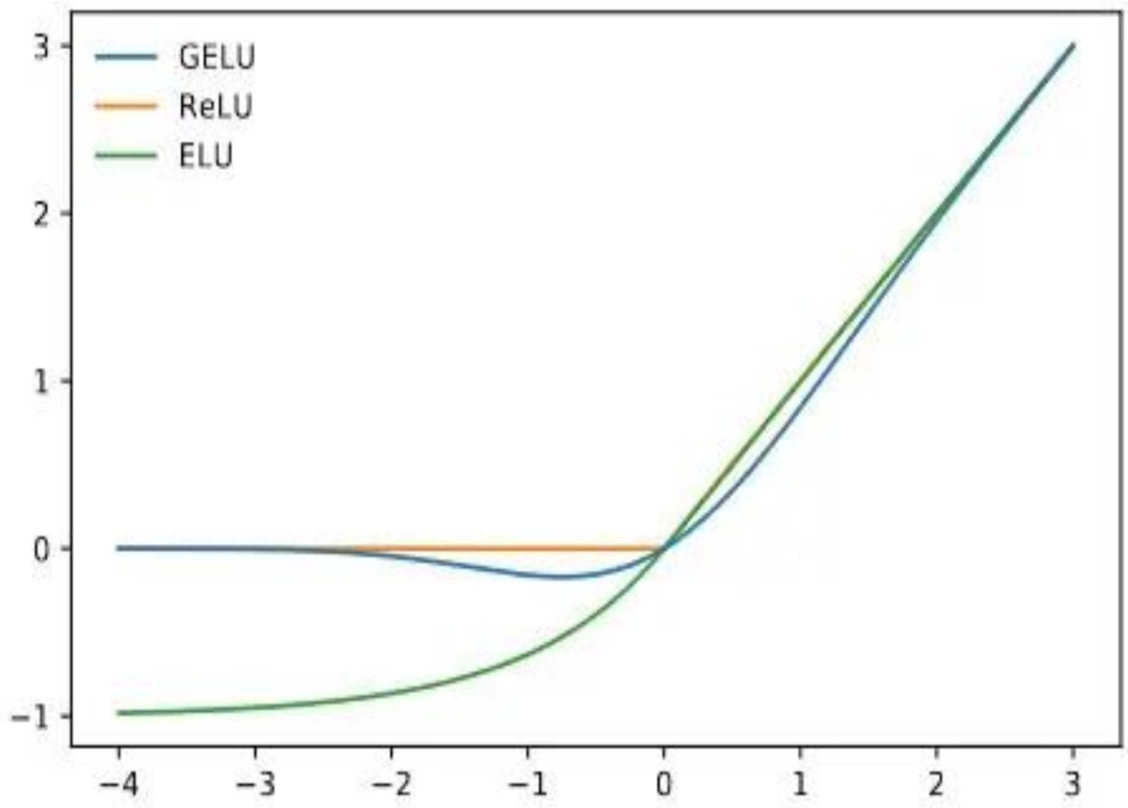


Derivative of ELU

Djork-Arné Clevert, Thomas Unterthiner, Sepp Hochreiter Fast and Accurate Deep Network Learning by Exponential Linear Units

Gaussian ELU (GELU)

$$GELU(x) = 0.5x(1 + \tanh(\sqrt{2/\pi}\,(x + 0.044715x^3))$$



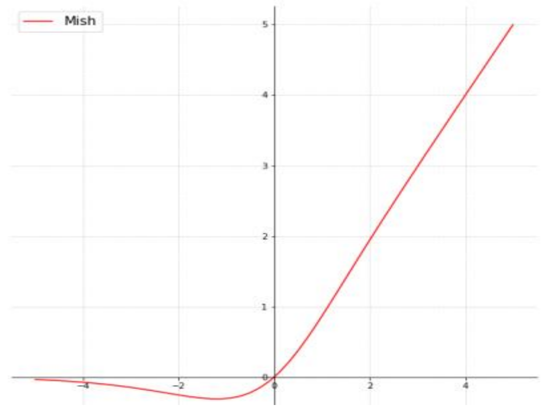
Hendrycks, Dan, and Kevin Gimpel. "Gaussian error linear units (gelus)." *arXiv preprint arXiv:1606.08415* (2016).

Recent trend: Hybrid Activation functions

- Mish Activation function

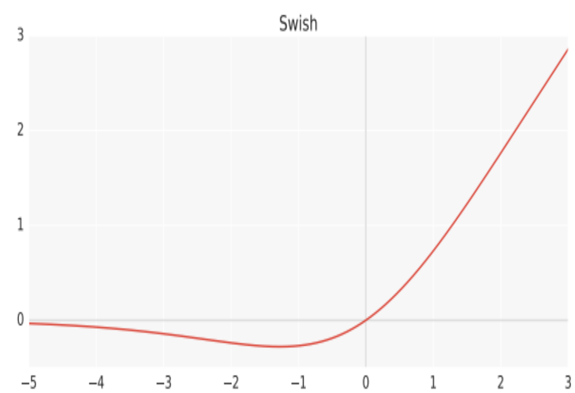
$$f(x) = x \cdot tanh(\varsigma(x))$$

$$\varsigma(x) = \ln(1 + e^x)$$



- Swish Activation function

$$f(x) = x \cdot sigmoid(x)$$

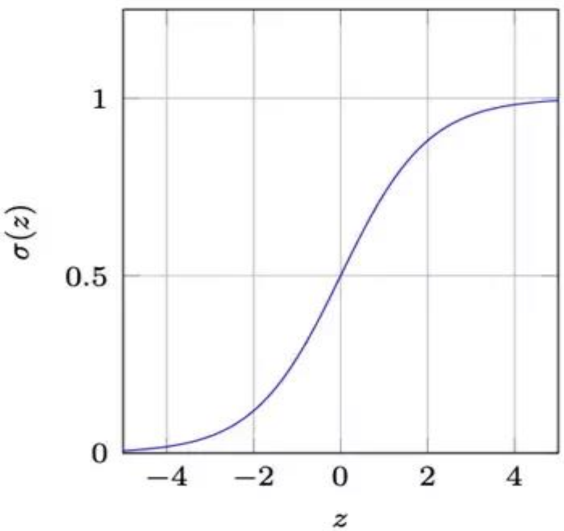


Misra, Diganta. "Mish: A Self Regularized Non-Monotonic Neural Activation Function." *arXiv preprint arXiv:1908.08681* (2019).

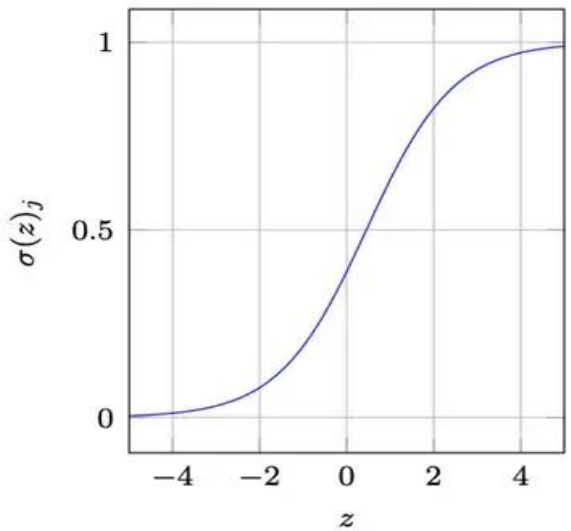
Ramachandran, Prajit, Barret Zoph, and Quoc V. Le. "Swish: a self-gated activation function." arXiv preprint arXiv:1710.05941 7 (2017).

Activation function in the output layer

- We usually consider the following two activation function for the output layer:
  - Sigmoid : two (binary) classes
  - Softmax : more than two classes



(a) Sigmoid activation function.



(b) Softmax activation function.

How to choose an activation function?

- **Million-dollar question in deep learning**: how to actually choose the right activation function when training a neural network from scratch?
- Different activation functions have different advantages and disadvantages and depending on the type of the **neural network the outcome may be different.**
- Starting point can be to choose one of the ReLU-based activation functions (including ReLU itself)[1]
  - Increase the convergence speed.
- For **classification layer**:
  - binary classification then the sigmoid activation function is a good choice
  - For the multi-class classification softmax function is better as it will output probability representation for each class.

[1] Lederer, Johannes. "Activation functions in artificial neural networks: A systematic overview." *arXiv preprint arXiv:2101.09957* (2021).

What “same activation and gradient updating” really means

- Usually mean **consistency and compatibility between**:
  - the **activation function** used in forward propagation, and
  - how **gradients behave and are updated** during backpropagation.
- Why?:
  - **The activation function controls the gradient flow.**
  - **If gradients behave badly,** training becomes slow, unstable, or impossible.

What can go wrong?

If derivatives are **too small** → **vanishing gradients**

If derivatives are **too large** → **exploding gradients**

Where it helps and how?

- Avoiding biased or dead update:
  - If activation and gradient update are mismatched:
    - weights in early layers update **too slowly**
    - some neurons **never activate again** (dead neurons)
    - learning focuses only on top layers
  - This leads to:
    - slow convergence
    - poor generalization
    - wasted model capacity

Initialization & activation work together

- **Same-scale activations → same-scale gradients**
  - Efficient learning requires:
    - activations have **zero mean**
    - controlled variance
    - gradients have similar scale across layers

- Why these pairings matter:

Activation	Initialization	Reason
Sigmoid / Tanh	Xavier	Keeps variance stable
ReLU / Leaky ReLU	He	Compensates for half-zero outputs

- If you mismatch them:
  - variance explodes or collapses
  - gradients become unstable

Consistent activation behavior → consistent gradient scale → **one learning rate works well**.

Information preservation perspective (important intuition)

- Think of training as **information flowing forward and backward**.
  - Forward pass: data → features
  - Backward pass: error → credit assignment

- Good activation functions:
  - preserve information forward
  - preserve learning signal backward

- Bad ones:
  - squash information
  - block gradients

- Efficient training requires **symmetry in information flow**.

[1] Lederer, Johannes. "Activation functions in artificial neural networks: A systematic overview." *arXiv preprint arXiv:2101.09957* (2021).

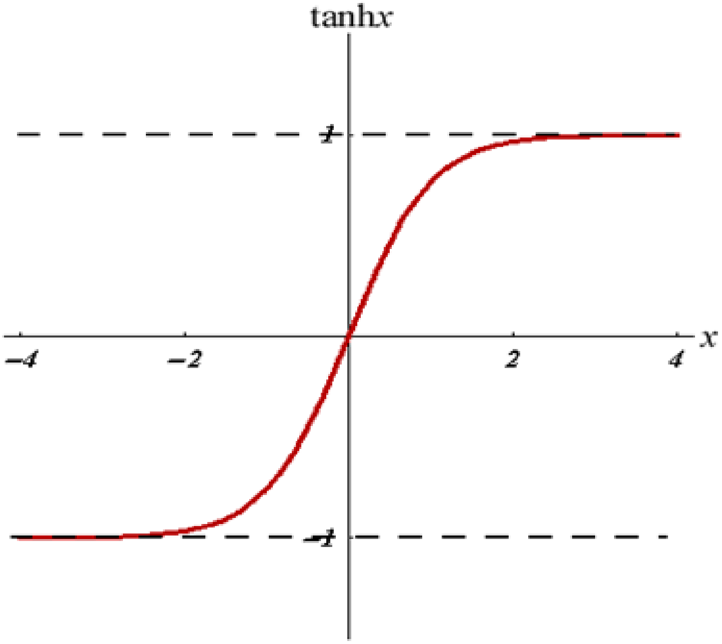
Regularization outline

- **L1 / L2 regularization**
- Early stopping
- Auxiliary classifiers
- Penalizing confident of output distributions and
- Dropout

- Batch normalization

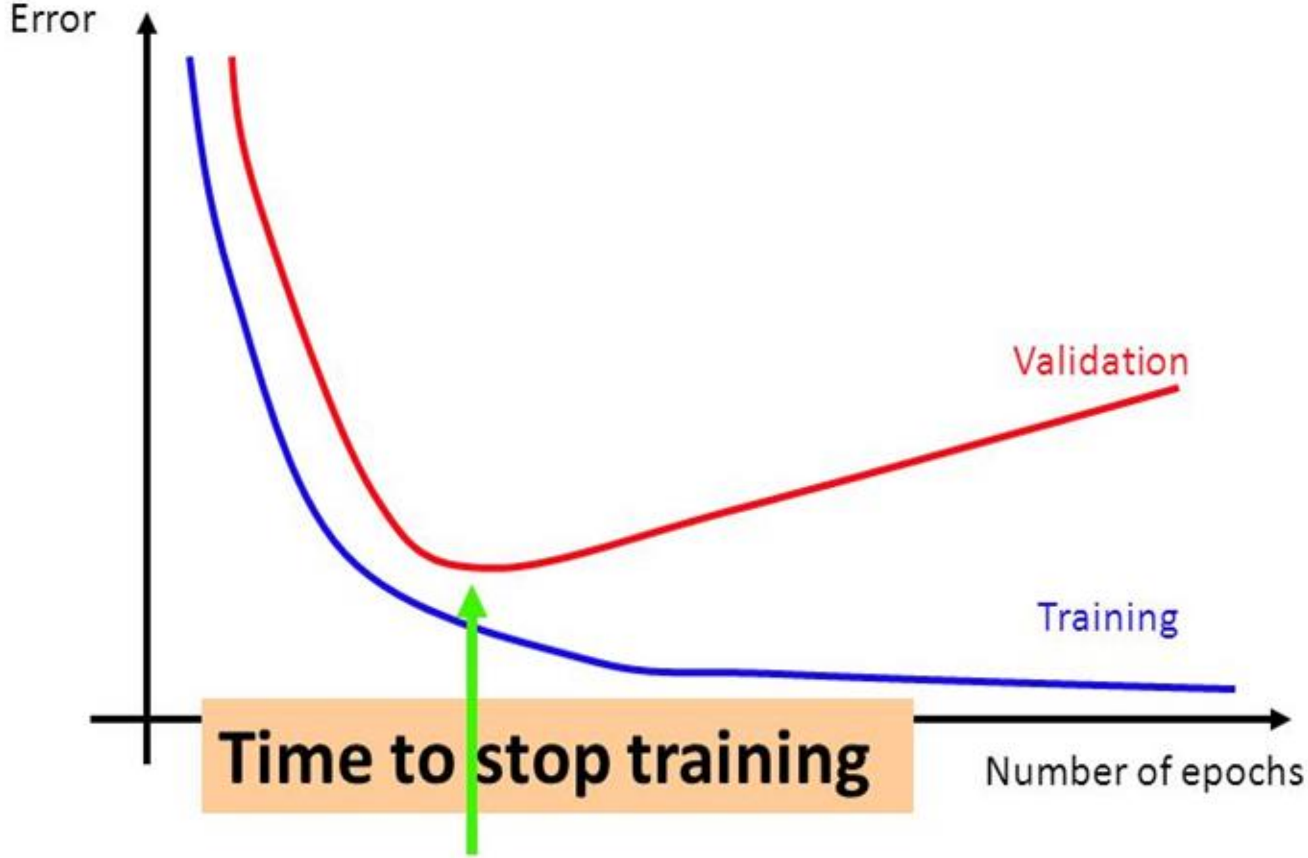
L1 / L2 regularization

- L1 encourages sparsity
- L2 discourages large weights
  - Gaussian prior on weight

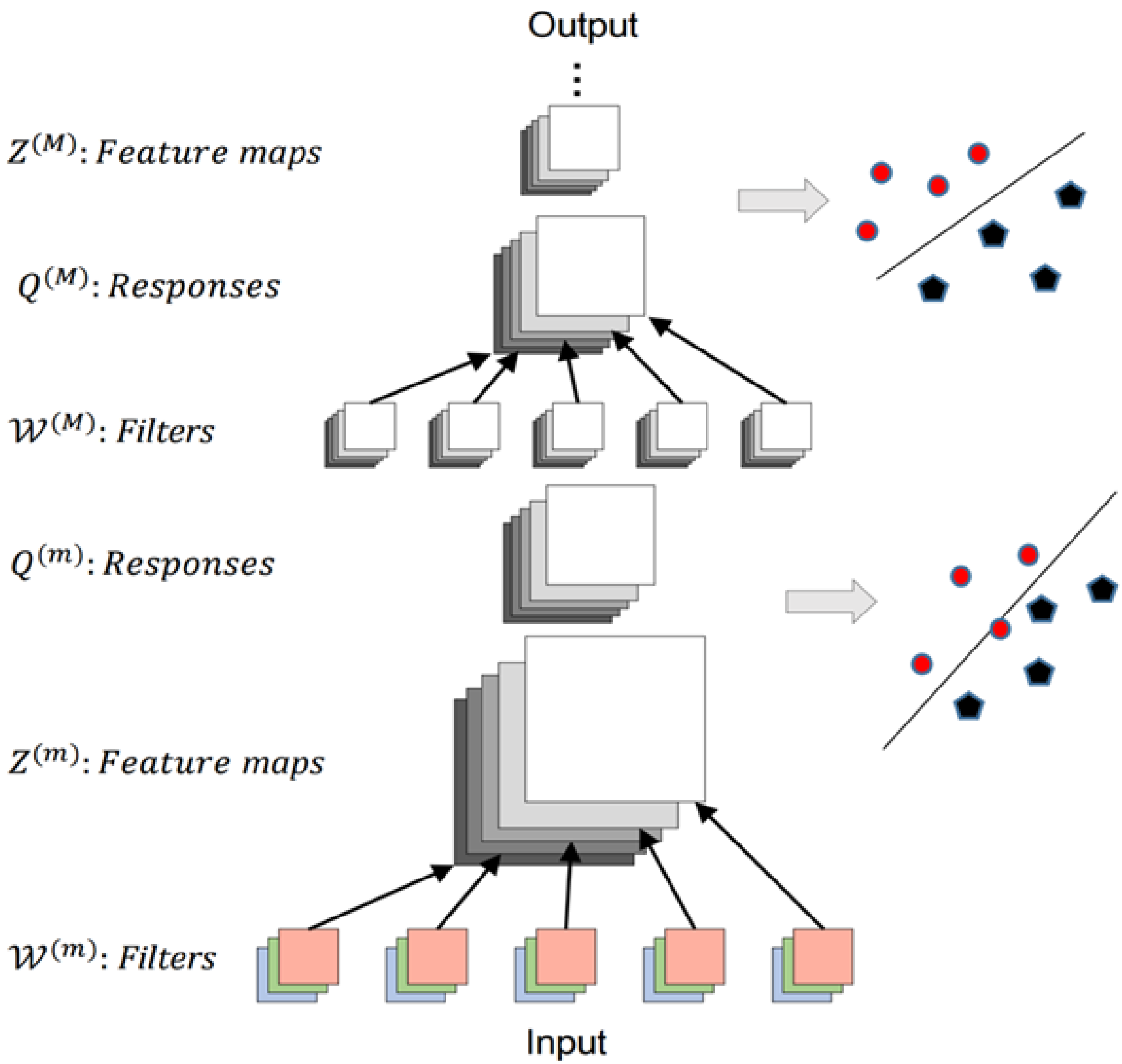


http://www.efunda.com/math/hyperbolic/images/tanh_plot.gif

Early Stopping



Auxiliary Classifiers



Lee et al. 2015. “Deeply Supervised Nets”

Penalizing outputs distributions

- Do **not allow the model to be overconfident**
- Prefer **smoother output distribution**
- Invariant to model parameterization

1. **Train towards smoother distribution**
2. Penalize entropy

True Label $\boxed{y_i} = [0, 0, 1, 0]$

Baseline $\boxed{u} = [0.25, 0.25, 0.25, 0.25]$

$$\hat{y}_i = (1 - \epsilon)\boxed{y_i} + \epsilon\boxed{u}$$

$\epsilon = 0.04$

Mixed Target

$$\hat{y}_i = [0.01, 0.01, 0.97, 0.01]$$

Szegedy et al. 2015 “Rethinking the Inception architecture for computer vision”

Penalizing confident distributions

$$y_i = [0, 0, 1, 0]$$

$$u = \boxed{[0.25, 0.25, 0.25, 0.25]}$$

$$\hat{y}_i = (1 - \epsilon)y_i + \epsilon u$$

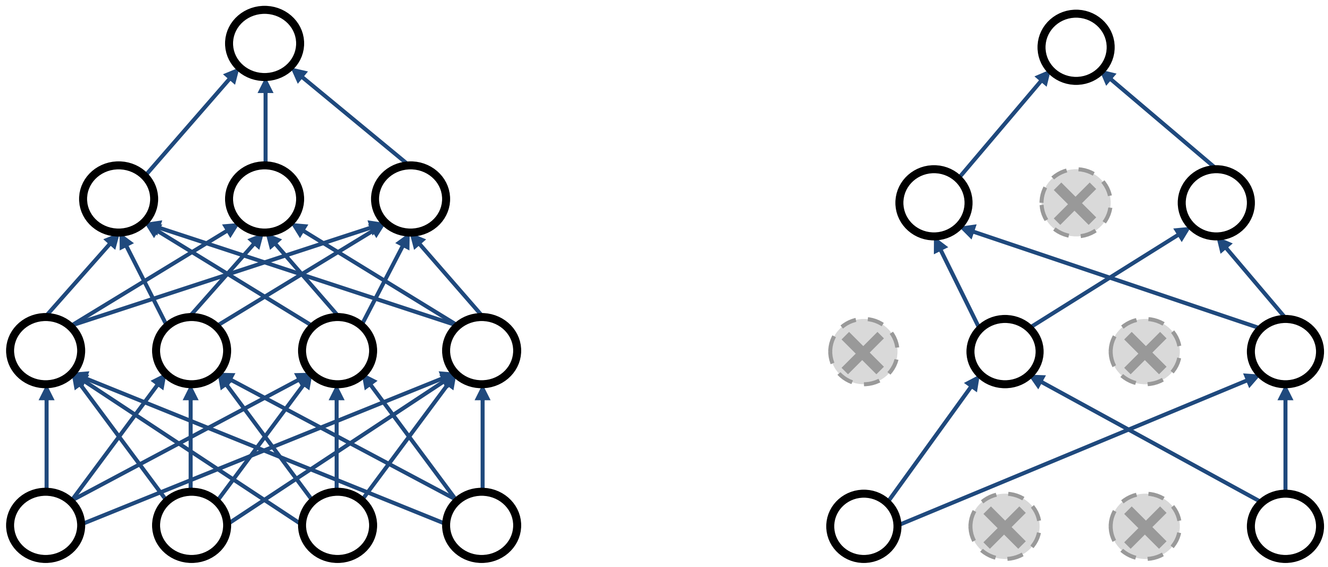
If $\epsilon = 0.04$

$$\hat{y}_i = [0.01, 0.01, 0.97, 0.01]$$

Szegedy et al. 2015 "Rethinking the Inception architecture for computer vision"

Regularization: Dropout

- In each forward pass, **randomly set some neurons to zero**
- Probability of dropping is **a hyperparameter**; 0.5 is common



Srivastava et al, “Dropout: A simple way to prevent neural networks from overfitting”, JMLR 2014

Regularization: Dropout

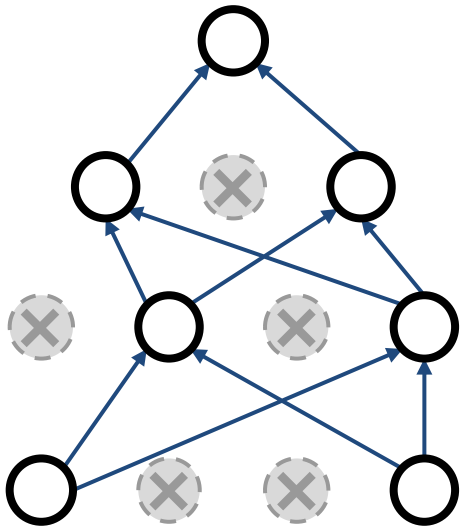
```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)
```

Example forward pass with a 3-layer network using dropout

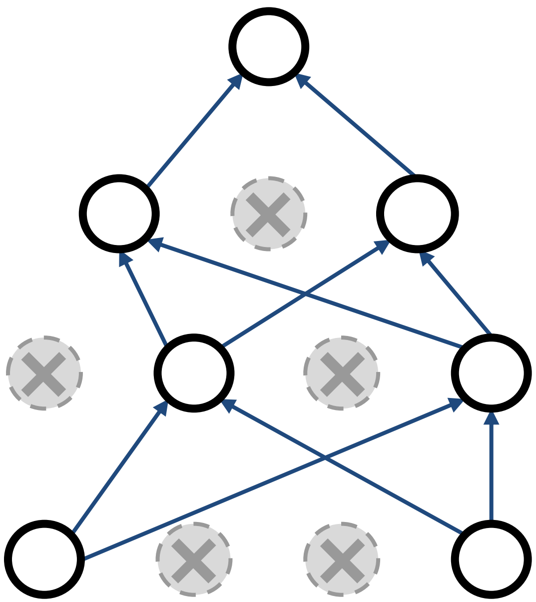


Regularization: Dropout

How can this possibly be a good idea?

Forces the network to have a redundant representation;

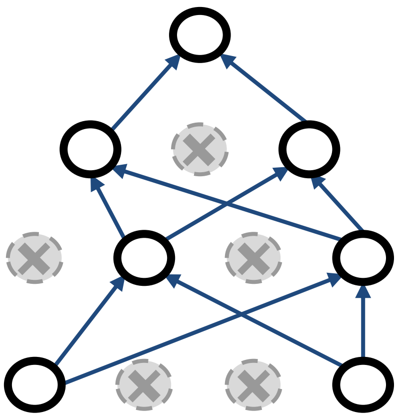
Prevents co-adaptation of features



Regularization: Dropout

How can this possibly be a good idea?

- Another interpretation:
  - Dropout is training a large **ensemble** of models (that share parameters).

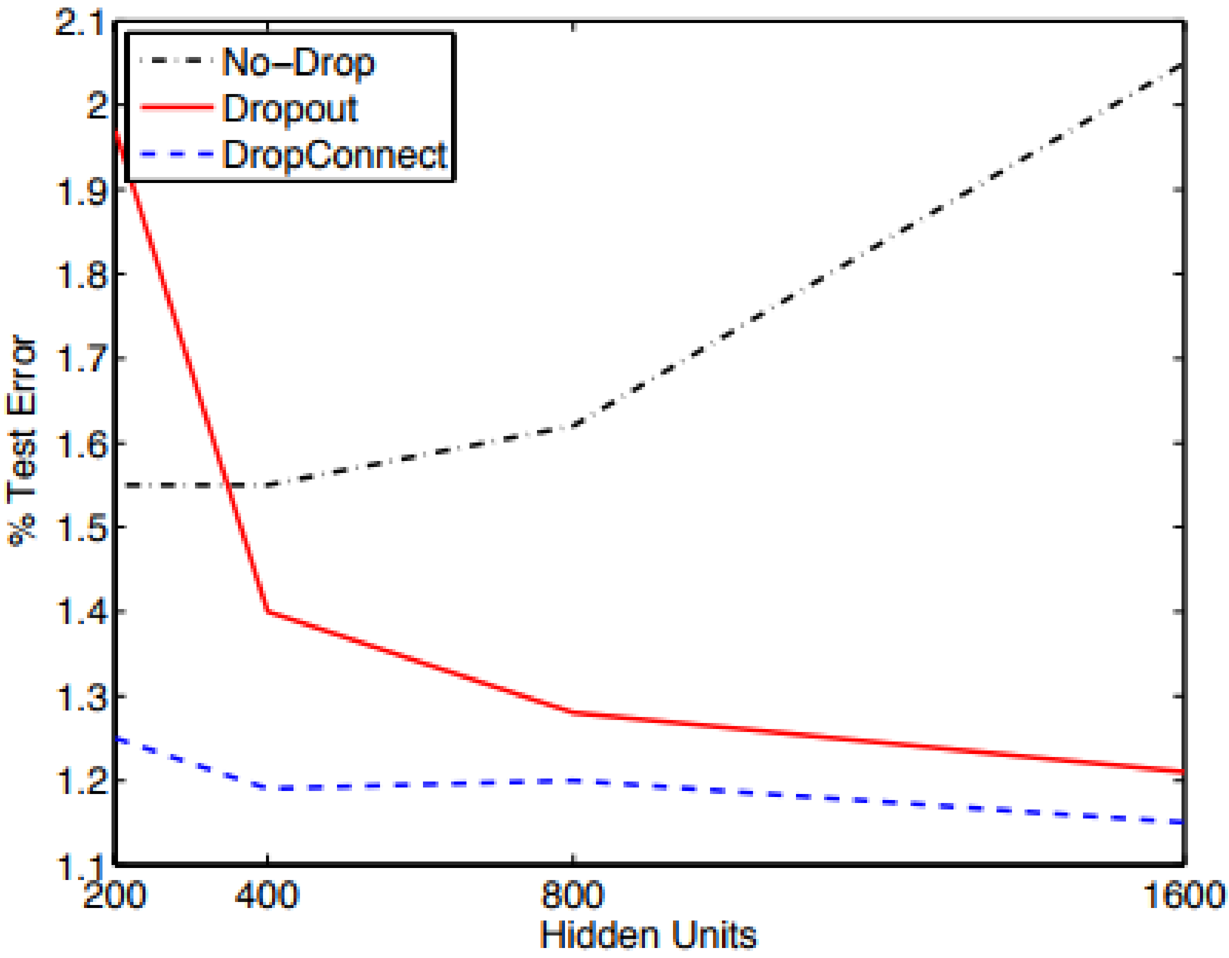


  - Each binary mask is for one model
  - An FC layer with 4096 units has

$2^{4096} \sim 10^{1233}$ possible masks!

Only $\sim 10^{82}$ atoms in the universe...

Drop-connection



Wan, Li, et al. "Regularization of neural networks using dropconnect." *Proceedings of the 30th international conference on machine learning (ICML-13)*. 2013.

Internal Covariate Shifting

- Distribution of inputs to a **layer is changing during training**

- Difficult to train, solutions:
  - careful initialization
  - smaller learning rate

- Easier if **distribution of inputs stayed the same**

- How to **enforce same distribution?**

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

Fighting with internal covariate shift

- Whitening would be a good first step
  - Would **remove nasty correlations**

- Problems with whitening?

  - Slow (have to do PCA for every layer)
  - **Cannot backprop through whitening**

- What is the best alternative?

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

Batch Normalization(BN)

- Calculate basic statistics per batch to make mean $\mu = 0$ and standard deviation $\sigma = 1$
- **Doesn't eliminate correlations**
- Fast and **can backprop through it**

- How to compute the statistics?
  - Going over the entire dataset is too slow
  - Idea: the **batch is an approximation** of the dataset
  - Compute **statistics over the batch**

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

Mean $\boxed{\mu_{\mathcal{B}}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} \boxed{x_i}$ Batch size

Variance $\boxed{\sigma_{\mathcal{B}}^2} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(\boxed{x_i} - \boxed{\mu_{\mathcal{B}}})^2$

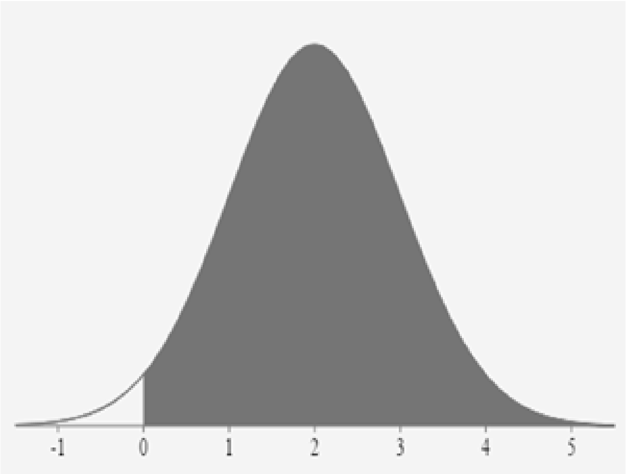
Normalize $\widehat{x}_i \leftarrow \frac{\boxed{x_i} - \boxed{\mu_{\mathcal{B}}}}{\sqrt{\boxed{\sigma_{\mathcal{B}}^2} + \epsilon}}$

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

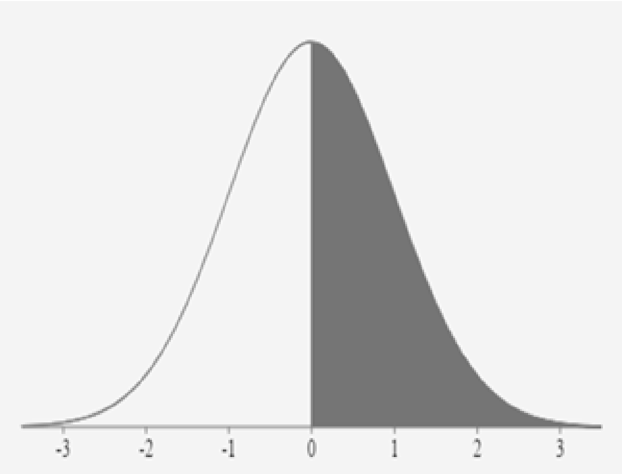
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

$$y_i \leftarrow \boxed{\gamma}\,\widehat{x}_i + \boxed{\beta} \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$$

Distribution of an activation before and after normalization



Before



After

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

Summary

- Data (inputs) preprocessing
- Initialization, and activation functions

  - **One-sentence takeaway**
    - **Using compatible activations and gradient updates is crucial because deep learning is a repeated product of derivatives**
      - **if activations distort gradient scale, learning either vanishes or explodes, making efficient training impossible.**

- Regularization methods including:
  - Regularization methods
  - Batch normalization

- What’s next?
  - Optimization methods and
  - Loss functions for training DNN models.

Appendix

Proof of He initialization

$$\mathbf{y}_l = W_l \mathbf{x}_l$$

W is a d-by-n matrix

$$\mathbf{y}_l = W_l \mathbf{x}_l$$

W is a d-by-n matrix

$$Var[y_l] = Var[w_l^1 x_l^1 + \cdots + w_l^n x_l^n]$$

$$\mathbf{y}_l = W_l \mathbf{x}_l$$

W is a d-by-n matrix

$$Var[y_l] = Var[w_l^1 x_l^1 + \cdots + w_l^n x_l^n]$$

w, x are both i.i.d. and independent of each other

$$\mathbf{y}_l = W_l \mathbf{x}_l$$

W is a d-by-n matrix

$$Var[y_l] = Var[w_l^1 x_l^1 + \cdots + w_l^n x_l^n]$$

w, x are both i.i.d. and independent of each other

$$Var[y_l] = n_l Var[w_l x_l]$$

$$Var[y_l] = n_l Var[w_l x_l]$$

$$Var[y_l] = n_l Var[w_l x_l]$$

Let w_l have zero mean

$$Var[y_l] = n_l Var[w_l x_l]$$

Let w_l have zero mean

$$Var[ab] = E[a^2]E[b^2] + E[a]^2 E[b]^2$$

$$Var[y_l] = n_l Var[w_l x_l]$$

Let w_l have zero mean

$$Var[ab] = E[a^2]E[b^2] + E[a]^2 E[b]^2$$

$$Var[y_l] = n_l(E[w_l^2]E[x_l^2] + E[w_l]^2 E[x_l]^2)$$

$$Var[y_l] = n_l Var[w_l x_l]$$

Let w_l have zero mean

$$Var[ab] = E[a^2]E[b^2] + E[a]^2 E[b]^2$$

$$Var[y_l] = n_l(E[w_l^2]E[x_l^2] + E[w_l]^2 E[x_l]^2)$$

$$Var[y_l] = n_l E[w_l^2]E[x_l^2]$$

$$Var[y_l] = n_l E[w_l^2] E[x_l^2]$$

$$Var[a] = E[(a - E[a])^2]$$

$$Var[y_l] = n_l E[w_l^2] E[x_l^2]$$

$$Var[a] = E[(a - E[a])^2]$$

$$Var[y_l] = n_l Var[w_l] E[x_l^2]$$

$$Var[y_l] = n_l Var[w_l] E[x_l^2]$$

$$x_l = \max(0, y_{l-1})$$

Let w_{l-1} have a zero-mean symmetric distribution

$$Var[y_l] = n_l Var[w_l] E[x_l^2]$$

$$x_l = \max(0, y_{l-1})$$

Let w_{l-1} have a zero-mean symmetric distribution

Then y_{l-1} also has a zero-mean symmetric distribution

$$Var[y_l] = n_l Var[w_l] E[x_l^2]$$

$$x_l = \max(0, y_{l-1})$$

Let w_{l-1} have a zero-mean symmetric distribution

Then y_{l-1} also has a zero-mean symmetric distribution

$$Var[y_{l-1}] = \int_{-\infty}^{0} y_{l-1}^2 p(y_{l-1}) dy_{l-1} + \int_{0}^{\infty} y_{l-1}^2 p(y_{l-1}) dy_{l-1}$$

$$Var[y_l] = n_l Var[w_l] E[x_l^2]$$

$$x_l = \max(0, y_{l-1})$$

Let w_{l-1} have a zero-mean symmetric distribution

Then y_{l-1} also has a zero-mean symmetric distribution

$$Var[y_{l-1}] = \int_{-\infty}^{0} y_{l-1}^2 p(y_{l-1}) dy_{l-1} + \int_{0}^{\infty} y_{l-1}^2 p(y_{l-1}) dy_{l-1}$$

$$Var[y_{l-1}] = 2\int_{0}^{\infty} y_{l-1}^2 p(y_{l-1}) dy_{l-1}$$

$$Var[y_{l-1}] = 2 \int_0^{\infty} y_{l-1}^2 p(y_{l-1}) dy_{l-1}$$

$$Var[y_{l-1}] = 2\int_0^\infty y_{l-1}^2 p(y_{l-1}) dy_{l-1}$$

$$Var[y_{l-1}] = 2\int_0^\infty x_l^2 p(x_l) dx_l$$

$$Var[y_{l-1}] = 2\int_0^\infty y_{l-1}^2 p(y_{l-1}) dy_{l-1}$$

$$Var[y_{l-1}] = 2\int_0^\infty x_l^2 p(x_l) dx_l$$

$$\frac{1}{2} Var[y_{l-1}] = E[x_l^2]$$

$$Var[y_{l-1}] = 2\int_0^\infty y_{l-1}^2 p(y_{l-1}) dy_{l-1}$$

$$Var[y_{l-1}] = 2\int_0^\infty x_l^2 p(x_l) dx_l$$

$$\frac{1}{2} Var[y_{l-1}] = E[x_l^2]$$

$$Var[y_l] = n_l Var[w_l] E[x_l^2]$$

$$Var[y_{l-1}] = 2\int_0^\infty y_{l-1}^2 p(y_{l-1}) dy_{l-1}$$

$$Var[y_{l-1}] = 2\int_0^\infty x_l^2 p(x_l) dx_l$$

$$\frac{1}{2} Var[y_{l-1}] = E[x_l^2]$$

$$Var[y_l] = n_l Var[w_l] E[x_l^2]$$

$$Var[y_l] = \frac{1}{2} n_l Var[w_l] Var[y_{l-1}]$$

$$Var[y_l] = \frac{1}{2} n_l Var[w_l] Var[y_{l-1}]$$

$$Var[y_l] = \frac{1}{2} n_l Var[w_l] Var[y_{l-1}]$$

$$Var[y_L] = Var[y_1] \prod_{l=2}^{L} \left(\frac{1}{2} n_l Var[w_l] \right)$$

$$Var[y_l] = \frac{1}{2} n_l Var[w_l] Var[y_{l-1}]$$

$$Var[y_L] = Var[y_1] \prod_{l=2}^{L} \left(\frac{1}{2} n_l Var[w_l] \right)$$

$$Var[y_L] = Var[y_1]$$

$$Var[y_l] = \frac{1}{2} n_l Var[w_l] Var[y_{l-1}]$$

$$Var[y_L] = Var[y_1] \prod_{l=2}^{L} \left(\frac{1}{2} n_l Var[w_l] \right)$$

$$Var[y_L] = Var[y_1]$$

$$\frac{1}{2} n_l Var[w_l] = 1$$

$$\frac{1}{2} n_l Var[w_l] = 1$$

$$\frac{1}{2} n_l Var[w_l] = 1$$

$$w_l \sim \mathcal{N}\left(0, \frac{2}{n_l}\right)$$