ENGR-E 533 “Deep Learning Systems”
Lecture 06: Adult Optimization

Minje Kim
Department of Intelligent Systems Engineering
Email: minje@indiana.edu
Website: http://minjekim.com
Research Group: http://saige.sice.indiana.edu
Meeting Request: http://doodle.com/minje

INDIANA UNIVERSITY
SCHOOL OF INFORMATICS,
COMPUTING, AND ENGINEERING

Initialization Methods, Activation Functions, and Batch Normalization

Yes, they are related

Initialization

-Why random?

- I remember a fellow student who didn't randomly initialize in his machine learning class a while ago
- What he did was to try out a bunch of different random numbers and pick up the best one
  - Best in the sense of test accuracy
- He was still using some MATLAB functions that generate random numbers
  - Why is not the random initialization?
  - What would be the correct random initialization?
- Random initialization means you generate random numbers to initialize your parameters
  - Therefore, there could be some good choices and bad choices depending on your luck
  - You never know until you see the results
- Eventually there's a technique called ensemble, but for now let's just bear with this uncertainty
- Remaining questions
  - What would be the right p.d.f. to sample from
  - How do we define the parameters of the p.d.f.?

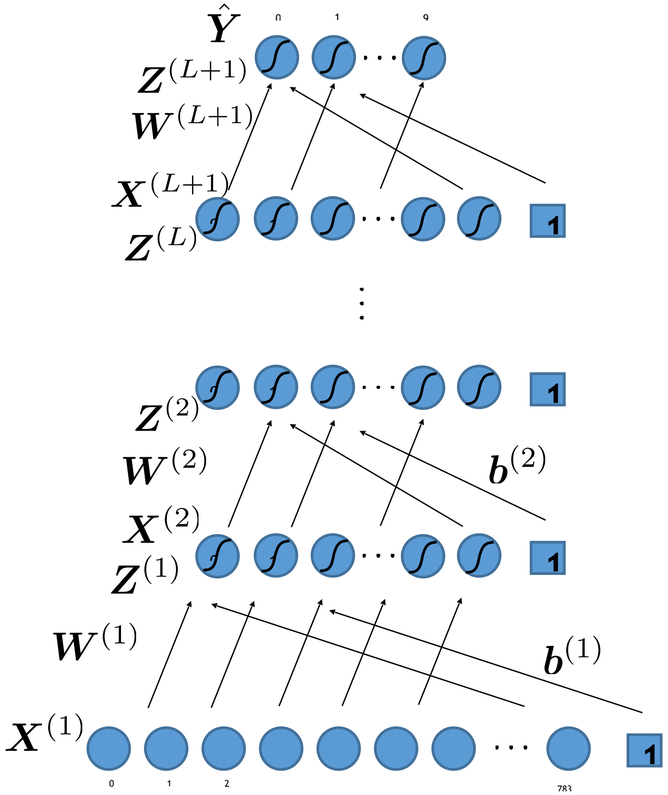
Initialization

-Choosing the best p.d.f.

- Optimal p.d.f. to sample your initial parameters from
  - Suppose the set of weights that give you the global optimum of the objective function
    - e.g. The weights that predict the test samples with the maximum possible accuracy
  - If we find a p.d.f. that's very similar to the sample distribution of these optimal weights
    - I have no theory behind this, but that would be the best one to start from
- Normally we should use some other distributions
  - Maybe some mismatching ones
  - What would be your choice?
  - Normal distribution $\mathcal{N}(\mu, \sigma)$ Many people also like to use uniform distributions, too
- What would be the mean and variance?
  - It might be safer to use zero mean
  - How about the variance?
  - Let's take a look at them
- Hold on.. You need to know which part of the network to look at:

$$\frac{\partial \mathcal{E}}{\partial \boldsymbol{W}^{(1)}} = \frac{\partial \mathcal{E}}{\partial \hat{\boldsymbol{Y}}} \frac{\partial \hat{\boldsymbol{Y}}}{\partial \boldsymbol{Z}^{(l+1)}} \prod_{l=1}^{L} \left[\frac{\partial \boldsymbol{Z}^{(l+1)}}{\partial \boldsymbol{X}^{(l+1)}} \frac{\partial \boldsymbol{X}^{(l+1)}}{\partial \boldsymbol{Z}^{(l)}} \right] \frac{\partial \boldsymbol{Z}^{(1)}}{\partial \boldsymbol{W}^{(1)}} = \left(\left(\boldsymbol{W}^{(2)\top} \left(\cdots \left(\boldsymbol{W}^{(L+1)\top} (\hat{\boldsymbol{Y}} - \boldsymbol{Y}) \right) \odot \sigma'(\boldsymbol{Z}^{(L)}) \cdots \right) \right) \odot \sigma'(\boldsymbol{Z}^{(1)}) \right) \boldsymbol{X}^{(1)\top}$$

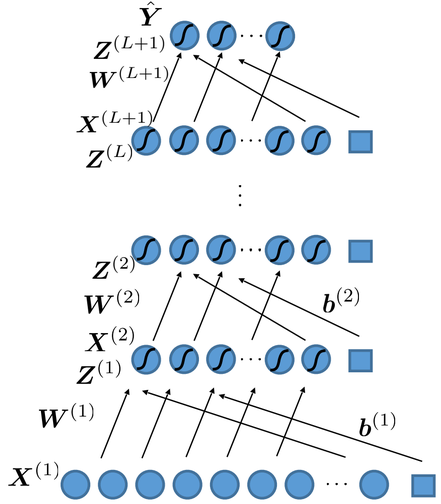
L speed bumps

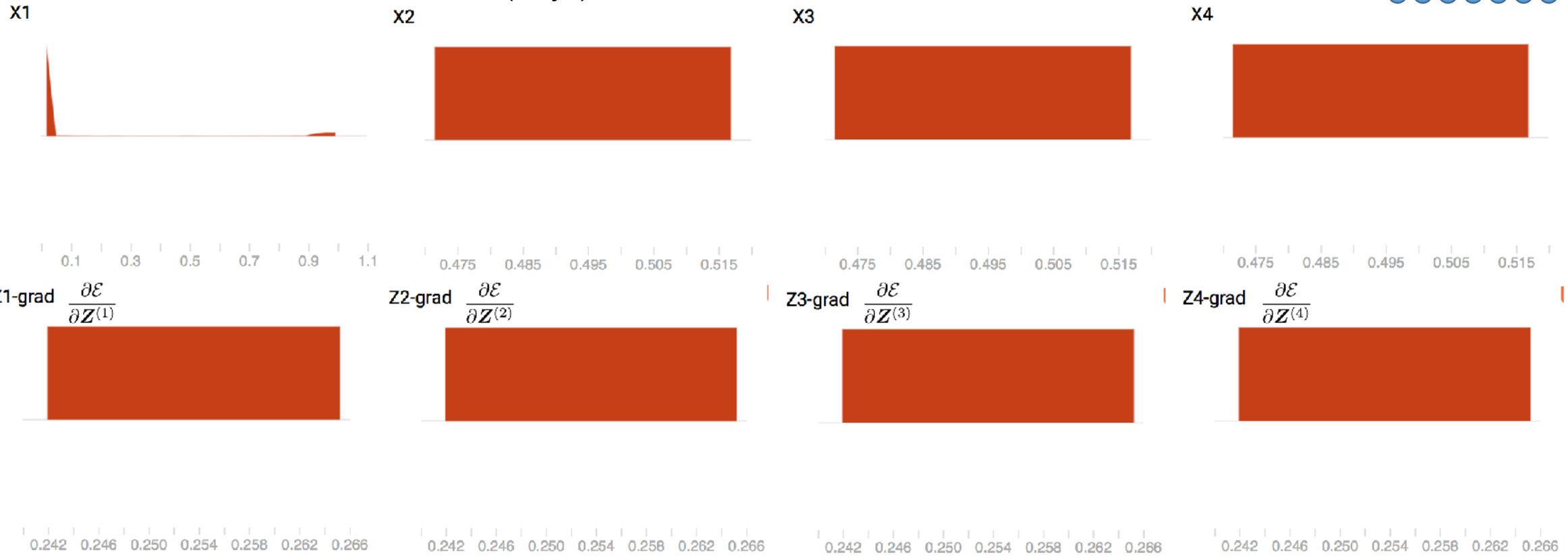


Initialization

-Choosing the best p.d.f.

- Too small variance $\sigma = 0.0001$
  - All the hidden unit activations (features) are centered around a value
    - Not healthy (features should be distinctive)
  - Even worse with the tanh activations (why?)

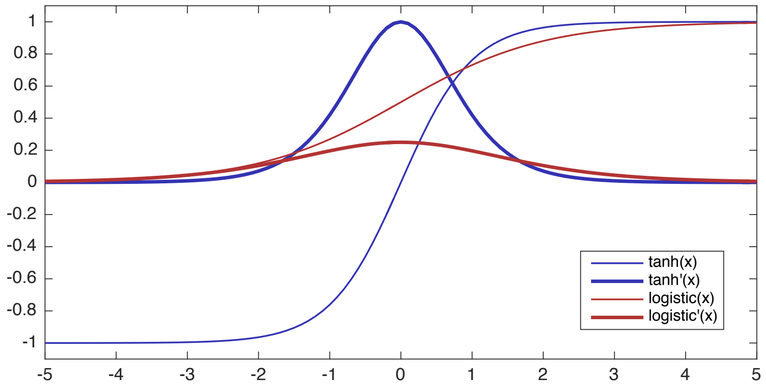


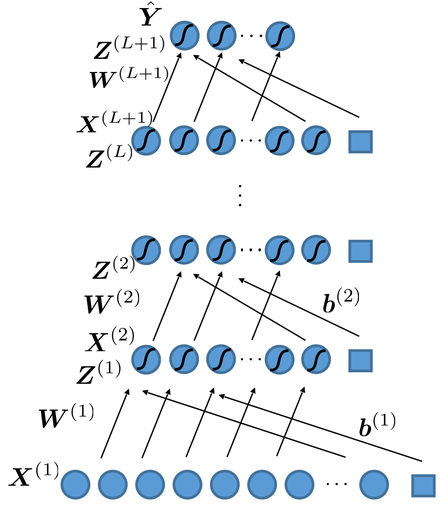


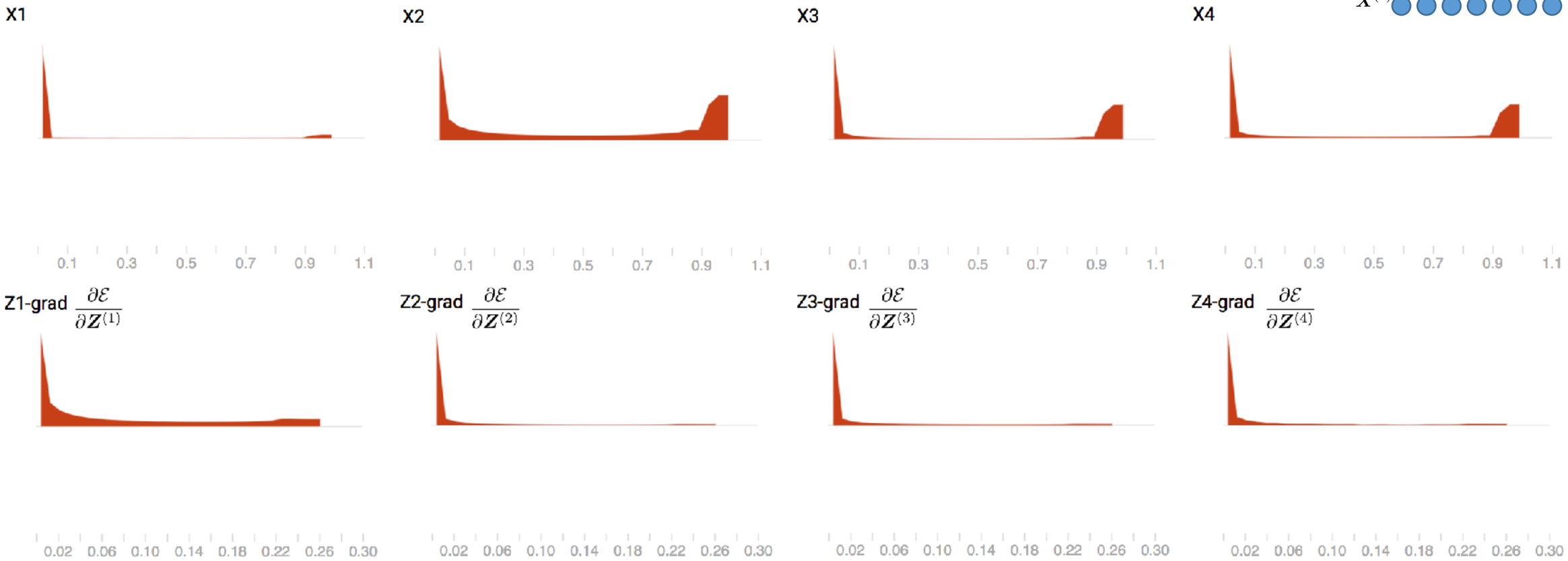
Initialization

-Choosing the best p.d.f.

- Too large variance $\sigma = 0.5$
  - Hidden unit outputs are too extreme (why?)
  - Gradient vanishes



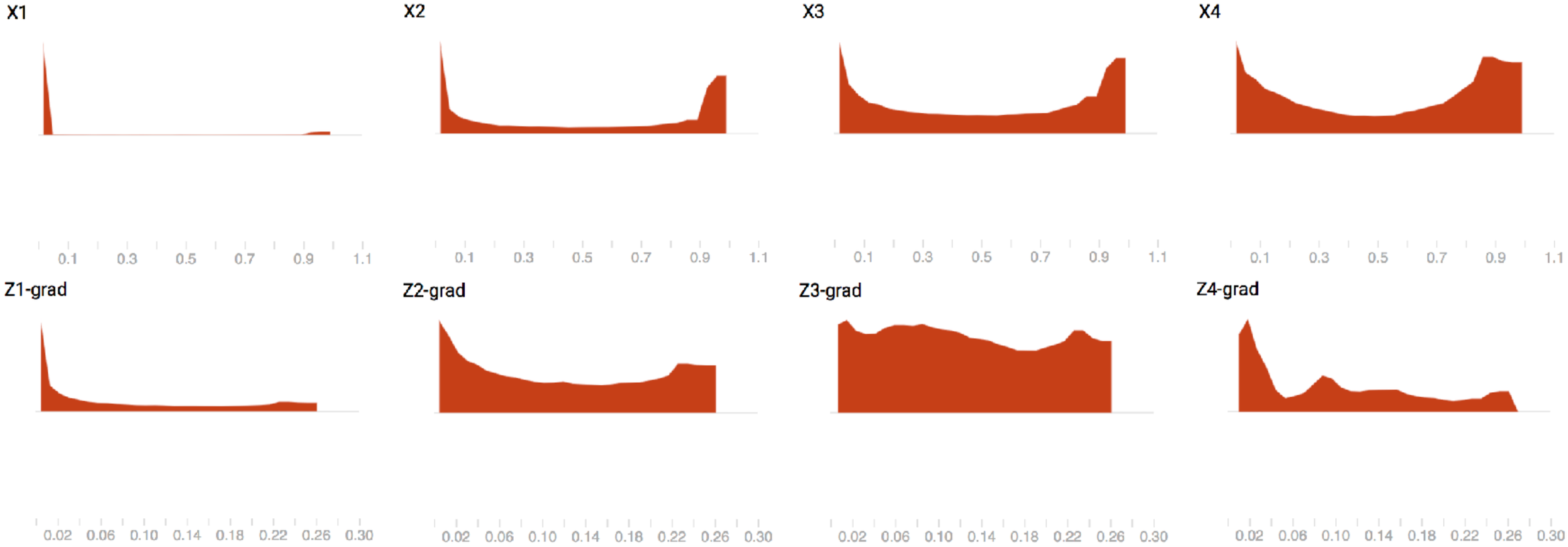




Initialization

-Dimension matters

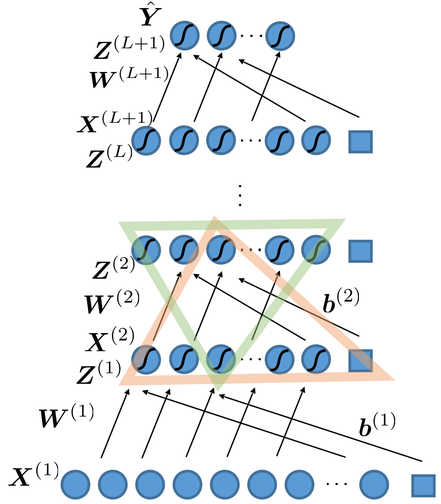
- Still too large variance $\sigma = 0.5$, but now are with only 50 hidden units per layer (previously it was 1,000)
  - Why does this happen?
  - $\text{var}(XY) = \text{var}(X)\text{var}(Y)$ (only if they are independent and zero mean)
  - So, what?



Frishman, Fred. "On the arithmetic means and variances of products and ratios of random variables." A Modern Course on Statistical Distributions in Scientific Work. Springer, Dordrecht, 1975. 401-406.

Initialization

-Xavier Initialization



- Variance of the linear combination $\boldsymbol{Z}_{i,t} = \sum_{j=1}^{N_j} \boldsymbol{W}_{i,j}\boldsymbol{X}_{j,t}$

$$\mathrm{var}(\boldsymbol{Z}_{i,t}) = N_j \mathrm{var}(\boldsymbol{W}_{i,j})\mathrm{var}(\boldsymbol{X}_{j,t})$$

  - To match the variance? $\mathrm{var}(\boldsymbol{W}_{i,j}) = \frac{1}{N_j}$

- We forgot about the other linear combination

$$\frac{\partial \mathcal{E}}{\partial \boldsymbol{W}^{(1)}} = \frac{\partial \mathcal{E}}{\partial \hat{\boldsymbol{Y}}} \frac{\partial \hat{\boldsymbol{Y}}}{\partial \boldsymbol{Z}^{(l+1)}} \prod_{l=1}^{L} \left[\frac{\partial \boldsymbol{Z}^{(l+1)}}{\partial \boldsymbol{X}^{(l+1)}} \frac{\partial \boldsymbol{X}^{(l+1)}}{\partial \boldsymbol{Z}^{(l)}} \right] \frac{\partial \boldsymbol{Z}^{(1)}}{\partial \boldsymbol{W}^{(1)}} = \left(\left(\boldsymbol{W}^{(2)\top} \left(\cdots \left(\boldsymbol{W}^{(L+1)\top} (\hat{\boldsymbol{Y}} - \boldsymbol{Y}) \right) \odot \sigma'(\boldsymbol{Z}^{(L)}) \cdots \right) \right) \odot \sigma'(\boldsymbol{Z}^{(1)}) \right) \boldsymbol{X}^{(1)\top}$$

$$\frac{\partial \mathcal{E}}{\partial \boldsymbol{X}^{(l)}} = \boldsymbol{W}^{(l)\top} \boldsymbol{\Delta}^{(l)}$$

$$\mathrm{var}\left(\left[\frac{\partial \mathcal{E}}{\partial \boldsymbol{X}^{(l)}} \right]_{j,t} \right) = \mathrm{var}\left(\sum_{i=1}^{N_i} \boldsymbol{W}_{i,j}^{(l)} \boldsymbol{\Delta}_{i,t}^{(l)} \right)$$

  - To match the variance? $\mathrm{var}(\boldsymbol{W}_{i,j}) = \frac{1}{N_i}$

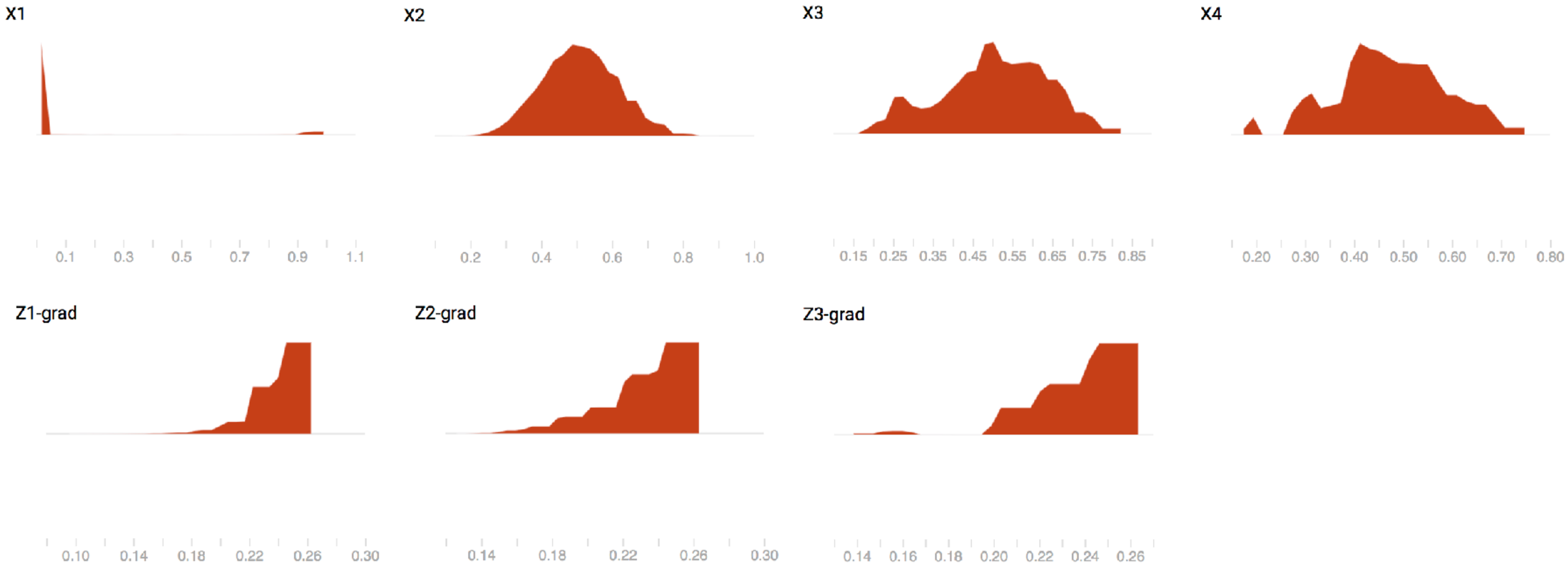
- Which one to choose?
  - Average number of units: $(N_j + N_i)/2$
  - Hence, $\mathrm{var}(\boldsymbol{W}_{i,j}) = \frac{2}{N_i + N_j}$

Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." Proceedings of the 13th International Conference on Artificial Intelligence and Statistics. 2010.

Initialization

-Xavier Initialization

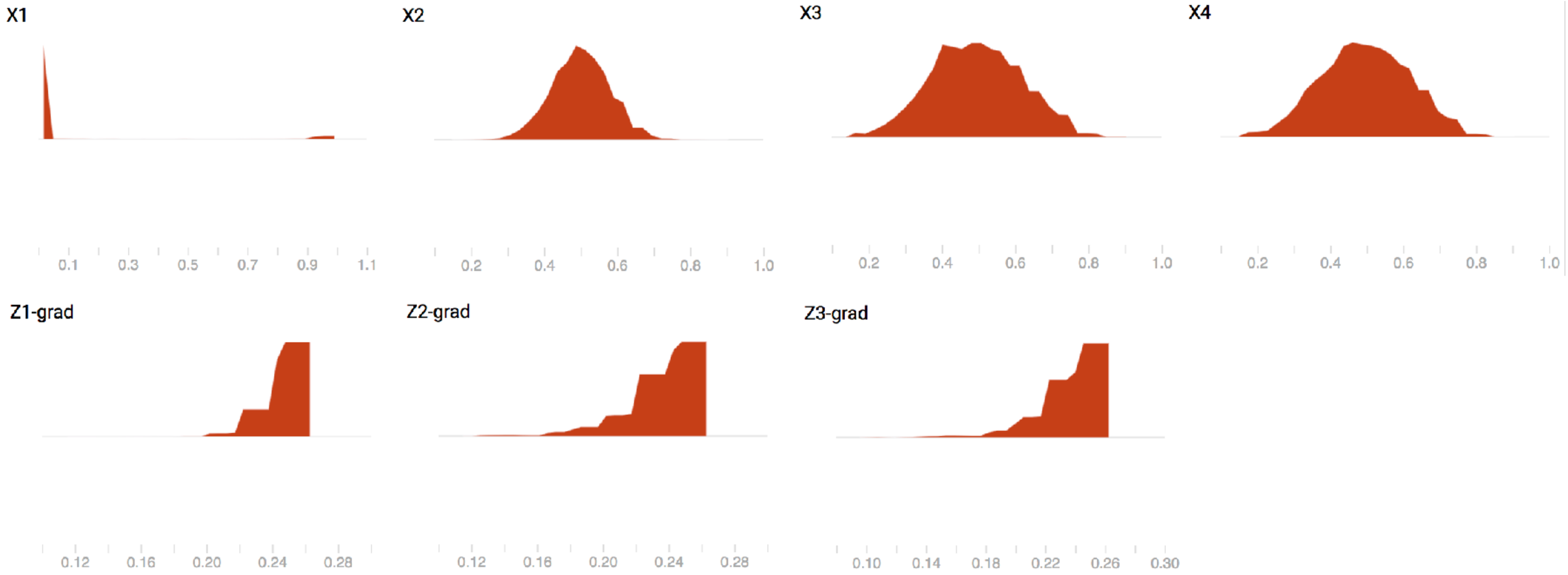
- Xavier initialization with 100 hidden units



Initialization

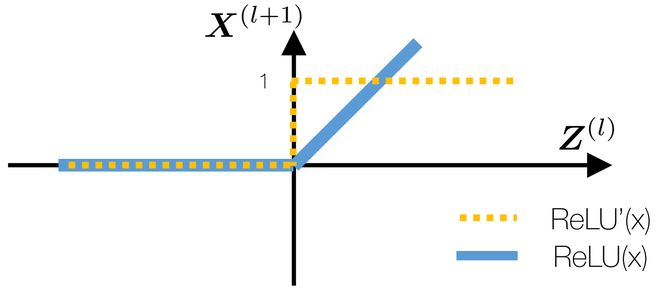
-Xavier Initialization

- Xavier initialization with 1000 hidden units
  - Properly adapts to the change of the network topology



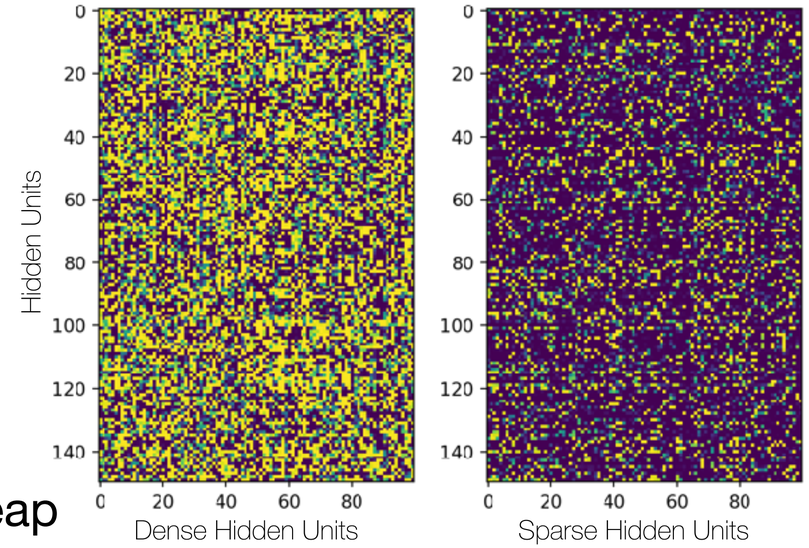
Activation Functions

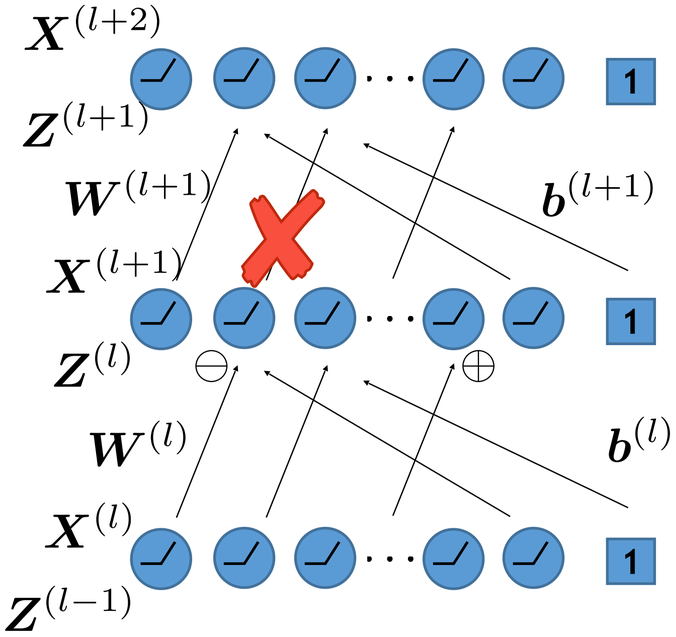
-Rectified Linear Units (ReLU)



- We all know that sigmoid functions are not a good choice for deep learning
  - Their derivatives block the flow of the BP error
- ReLU doesn't (entirely) block the BP error
  - It actually does block about half of them, but the other half survive
  - It actually blocks about half of the FP (forwardpropagation), too, but the other half survive
- Blocking FP path is actually good
  - A natural way to enforce sparsity
- Blocking BP path is NOT good
  - But we do have half the pathways survived

$$\frac{\partial \boldsymbol{X}_{i,t}^{(l+1)}}{\partial \boldsymbol{Z}_{i,t}^{(l)}} = \begin{cases} 1 & \boldsymbol{Z}_{i,t}^{(l)} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$





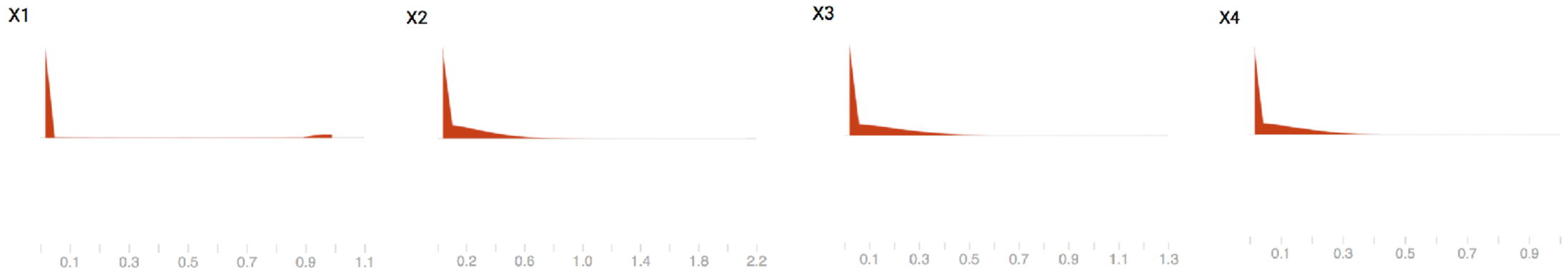
- Another thing to note is that ReLU is cheap to calculate both for FP and BP
- Let's see what happens

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks." Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. 2011.

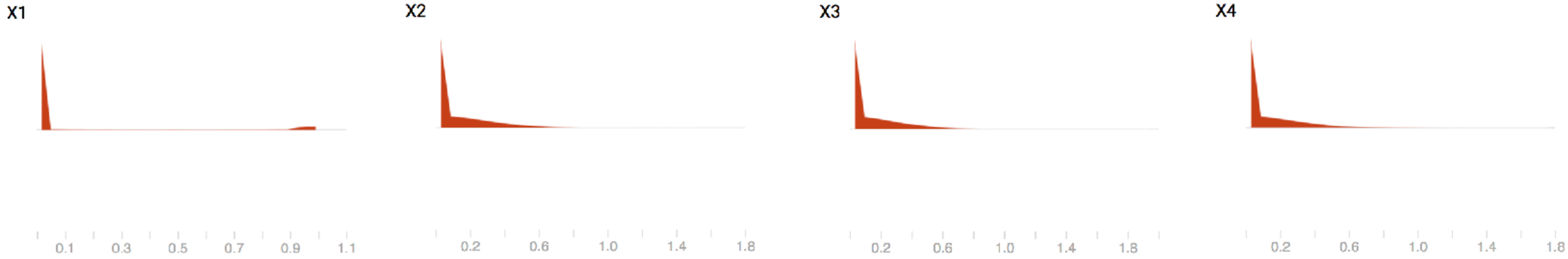
Activation Functions

-Rectified Linear Units (ReLU)

- ReLU with Xavier initialization (notice the different xtick ranges)



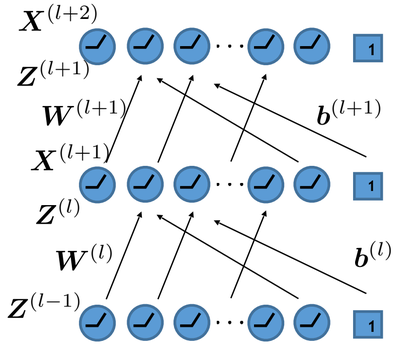
- ReLU with He initialization



He, Kaiming, et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." Proceedings of the IEEE international conference on computer vision. 2015.

Activation Functions

-He initialization for ReLU



- Variance of the linear combination $Z_{i,t}^{(l+1)} = \sum_{j=1}^{N_j} W_{i,j}^{(l+1)} X_{j,t}^{(l+1)}$

$$\mathrm{var}(Z_{i,t}^{(l+1)}) = N_j \mathrm{var}(W_{i,j}^{(l+1)})\mathrm{var}(X_{j,t}^{(l+1)}) + N_j \mathrm{var}(W_{i,j}^{(l+1)}) \left[E(X_{j,t}^{(l+1)})\right]^2 + N_j \mathrm{var}(X_{j,t}^{(l+1)}) \left[E(W_{i,j}^{(l+1)})\right]^2$$

This is correct only if your r.v.'s on the right hand side is centered

  - We have no control over the units, but can still center the weights $\therefore \left[E(X_{j,t})\right]^2 = 0$

$$\mathrm{var}(Z_{i,t}^{(l+1)}) = N_j \mathrm{var}(W_{i,j}^{(l+1)}) \left(\mathrm{var}(X_{j,t}^{(l+1)}) - \left[E(X_{j,t}^{(l+1)})\right]^2\right) = N_j \mathrm{var}(W_{i,j}^{(l+1)}) \left[E\left(X_{j,t}^{(l+1)^2}\right)\right] \stackrel{?}{=} \mathrm{var}(Z_{j,t}^{(l)})$$

  - We look for $\mathrm{var}(W_{i,j}^{(l+1)})$ that meets the final equation

- Let's see what happens in the lower layer
- Since we assume a centered symmetric distribution for $W_{i,j}^{(l)}$, should $P\left(Z_{i,j}^{(l)}\right)$ be, too

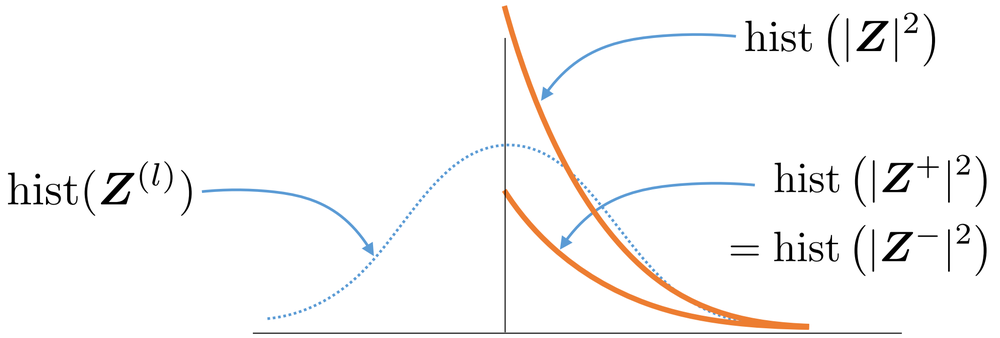
$$E\left(X_{j,t}^{(l+1)^2}\right) = E\left(\mathrm{ReLU}\left(Z_{j,t}^{-(l)}\right)^2\right)/2 + E\left(\mathrm{ReLU}\left(Z_{j,t}^{+(l)}\right)^2\right)/2$$

$$= E\left(\left(Z_{j,t}^{+(l)}\right)^2\right)/2 = E\left(Z_{j,t}^{(l)^2}\right)/2 = \mathrm{var}(Z_{j,t}^{(l)})/2$$

- Therefore,

$$\mathrm{var}(Z_{i,t}^{(l+1)}) = \frac{N_j}{2}\mathrm{var}(W_{i,j}^{(l+1)})\mathrm{var}(Z_{j,t}^{(l)}) \stackrel{?}{=} \mathrm{var}(Z_{j,t}^{(l)})$$

$$\mathrm{var}(W_{i,j}^{(l+1)}) = \frac{2}{N_j}$$

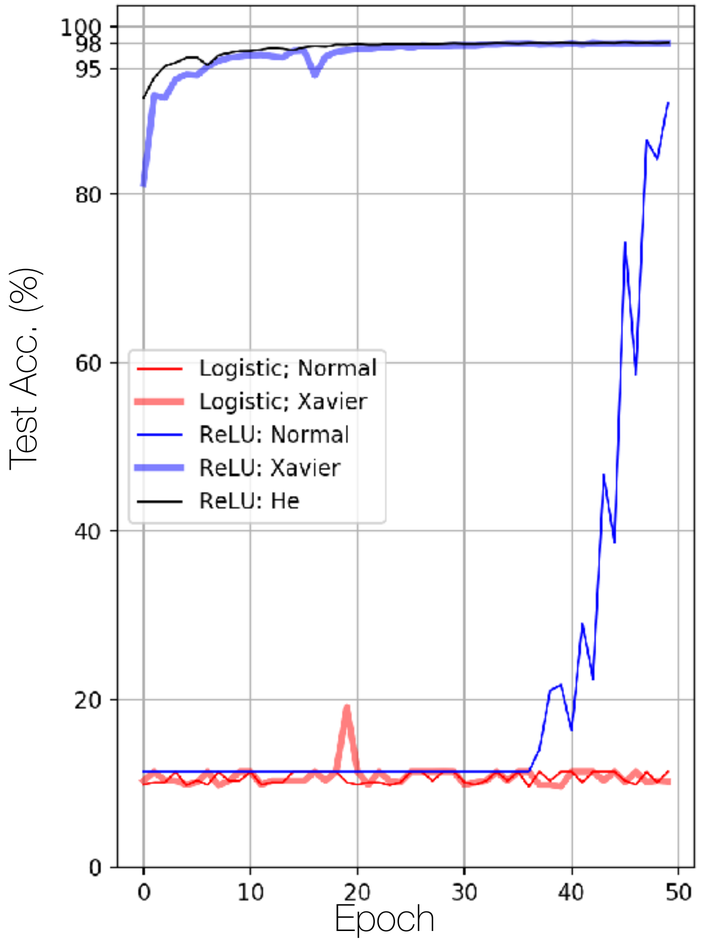


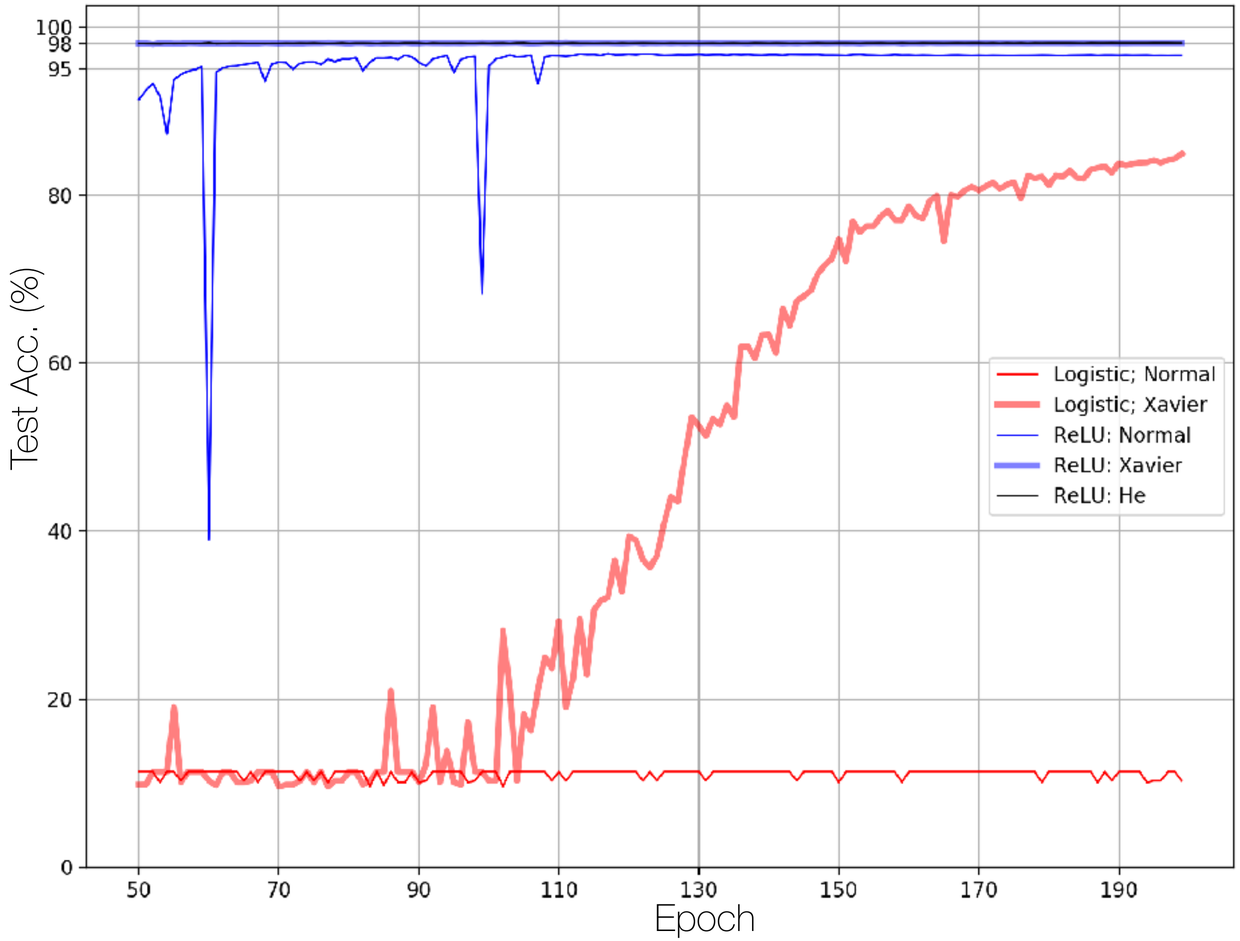
He, Kaiming, et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." Proceedings of the IEEE international conference on computer vision. 2015.

Activation Functions

-He initialization for ReLU

- For a 512X5 network for MNIST





Activation Functions

-The other activation functions

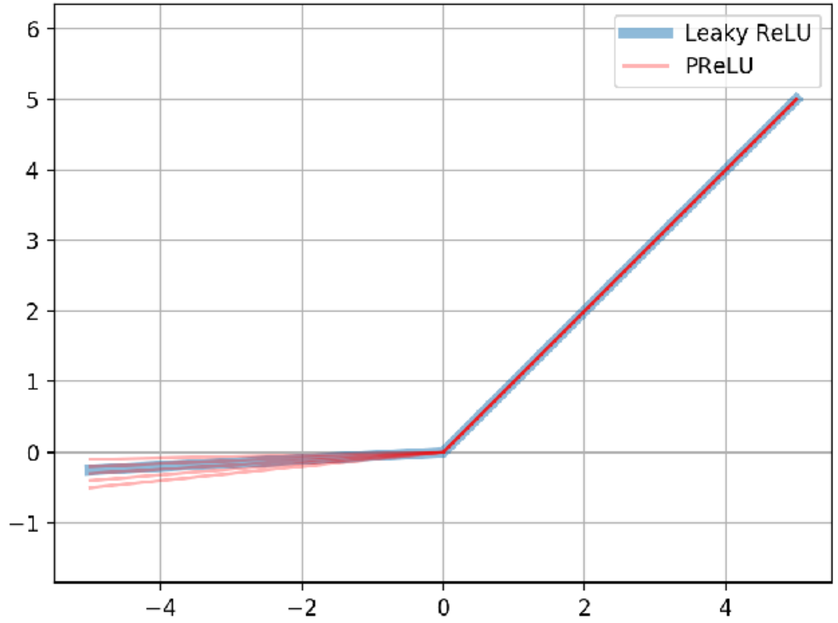
Leaky ReLU $f(x) = \max(x, 0) + a \min(0, x)$

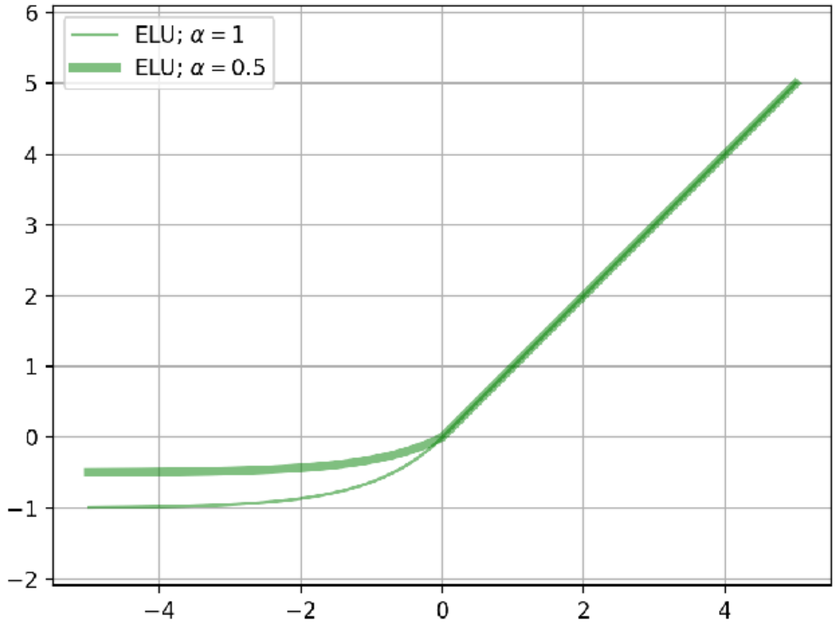
Parametric ReLU $f(x) = \max(x, 0) + a \min(0, x)$

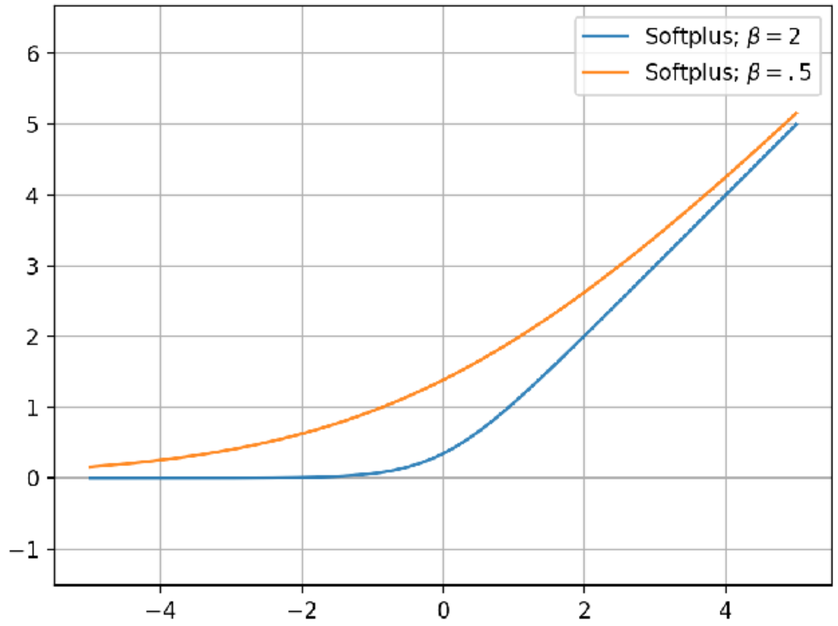
Softplus $f(x) = \frac{1}{\beta} \log(1 + \exp(\beta x))$

Exponential LU $f(x) = \max(x, 0) + \min(0, \alpha(\exp(x) - 1))$

You name it!



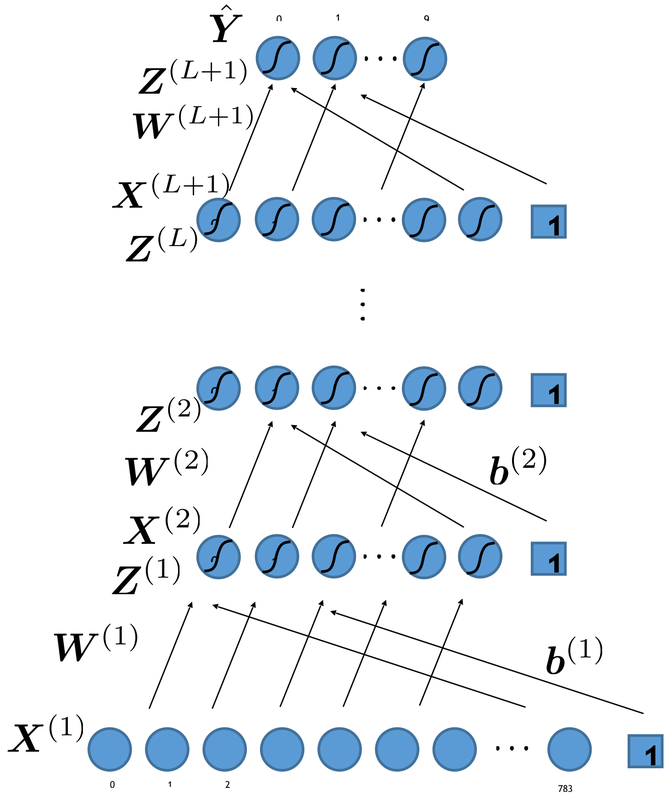




Batch Normalization

-A straightforward way to keep the distribution organized

- What is a batch?
- Let me do some calculation first
  - I've got a 1024X3 DNN for MNIST classification
  - What's the size of the input training matrix $\boldsymbol{X}^{(1)}$?
    - 784X55,000=43,120,000
    - In byte?
      + 784X55,000X32/8=172.48MB
  - What's the size of $\boldsymbol{X}^{(2)}$?
    - 1024X55,000X32/8=225.28MB
  - Wha'ts the size of $\boldsymbol{W}^{(2)}$?
    - 1024X1024X32/8=4.2MB
- These days this amount of calculation can be done in one shot in some good computers
  - But was too big back in the days
- Also, there are many datasets larger than MNIST out there
- Mini-batches is a compromised version of batch processing and SGD
- When the mini-batch size is 128: $\boldsymbol{X}^{(1)} \in \mathbb{R}^{784\times128}, \boldsymbol{X}^{(2)} \in \mathbb{R}^{1024\times128}, \cdots$



Ioffe, S. & Szegedy, C.. (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," International Conference on Machine Learning

Batch Normalization

-A straightforward way to keep the distribution organized

- So far we've seen that some wrong initialization can damage the convergence
  - The intermediate variables can have drastically different distributions across layers
    - Due to the wrong choice of initialization, and also because of a particular choice of activation functions
    - But it can happen anytime during training in deep learning
- Batch normalization: makes sure a variable (unit) follow the standard normal dist.

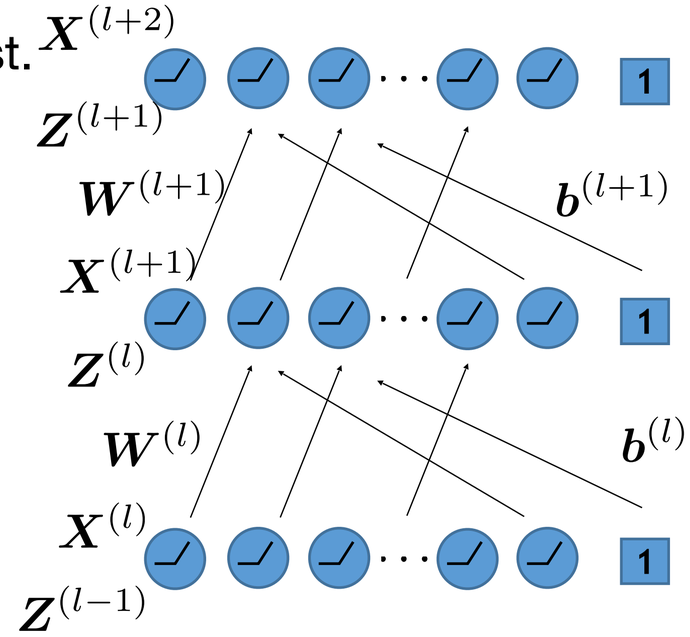
$$\boldsymbol{X}_{i,t}^{(l)} \sim [\text{Some distribution we don't like}] \xrightarrow{\text{Batch Normalization}} \overline{\boldsymbol{X}_{i,t}^{(l)}} \sim \mathcal{N}(0,1)$$

$t = [1, \cdots, N_b]$ is for the sample index for the batch

$b = [1, \cdots, B]$ denotes batches

$$\boldsymbol{\mu}_{i,b}^{(l)} = \frac{1}{N_b}\sum_{t=1}^{N_b} \boldsymbol{X}_{i,t}^{(l)} \qquad \boldsymbol{\sigma}_{i,b}^{(l)} = \frac{1}{N_b}\sum_{t=1}^{N_b}\left(\boldsymbol{X}_{i,t}^{(l)} - \boldsymbol{\mu}_{i,b}^{(l)}\right) \qquad \overline{\boldsymbol{X}_{i,t}^{(l)}} = \frac{\boldsymbol{X}_{i,t}^{(l)} - \boldsymbol{\mu}_{i,b}^{(l)}}{\boldsymbol{\sigma}_{i,b}^{(l)}}$$

  - Note that N_b is the number of samples in a (mini)batch



- This is a bit problematic
  - All input units of all layers are forced to produce the same sample distribution
  - Let's give them some flexibility $\widehat{\boldsymbol{X}_{i,t}^{(l)}} = \gamma_i^{(l)}\overline{\boldsymbol{X}_{i,t}^{(l)}} + \beta_i^{(l)}$

Note that this linear combination doesn't have the mini-batch index

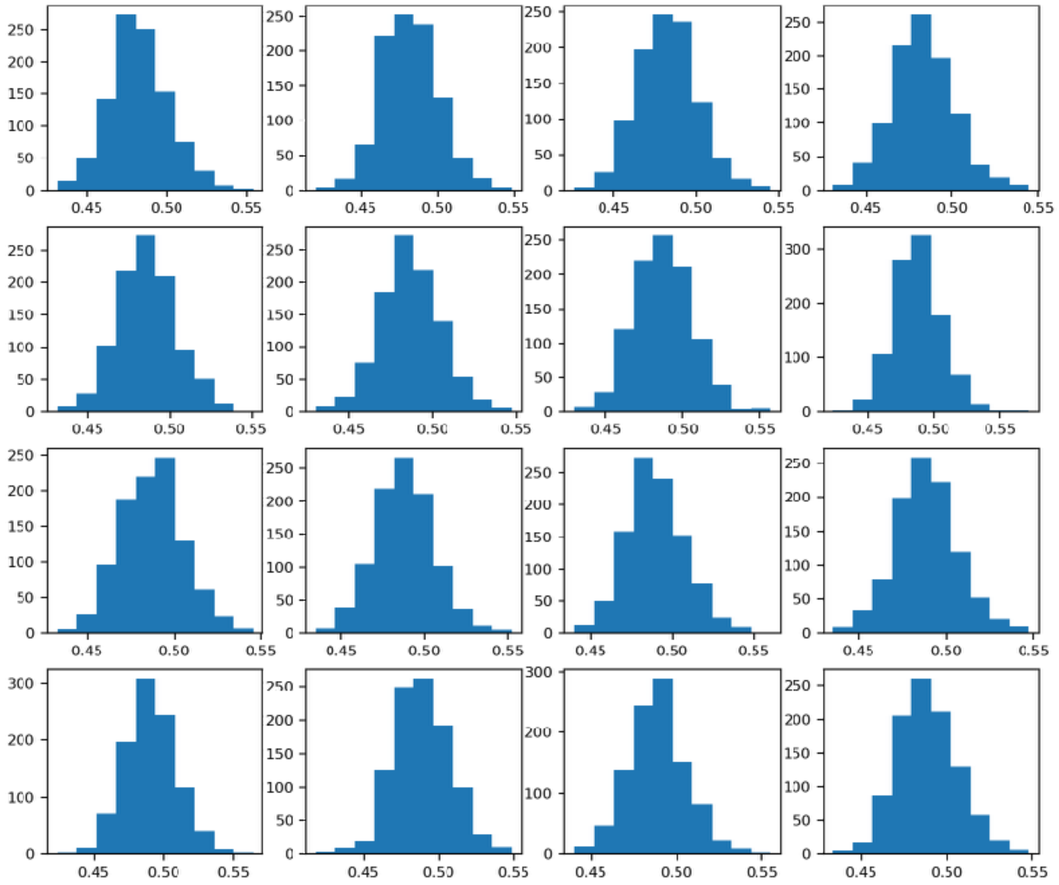
- Eventually,
  batch normalization standardizes mini-batch statistics, but keeps the empirical statistics of the entire dataset

Ioffe, S. & Szegedy, C.. (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," International Conference on Machine Learning

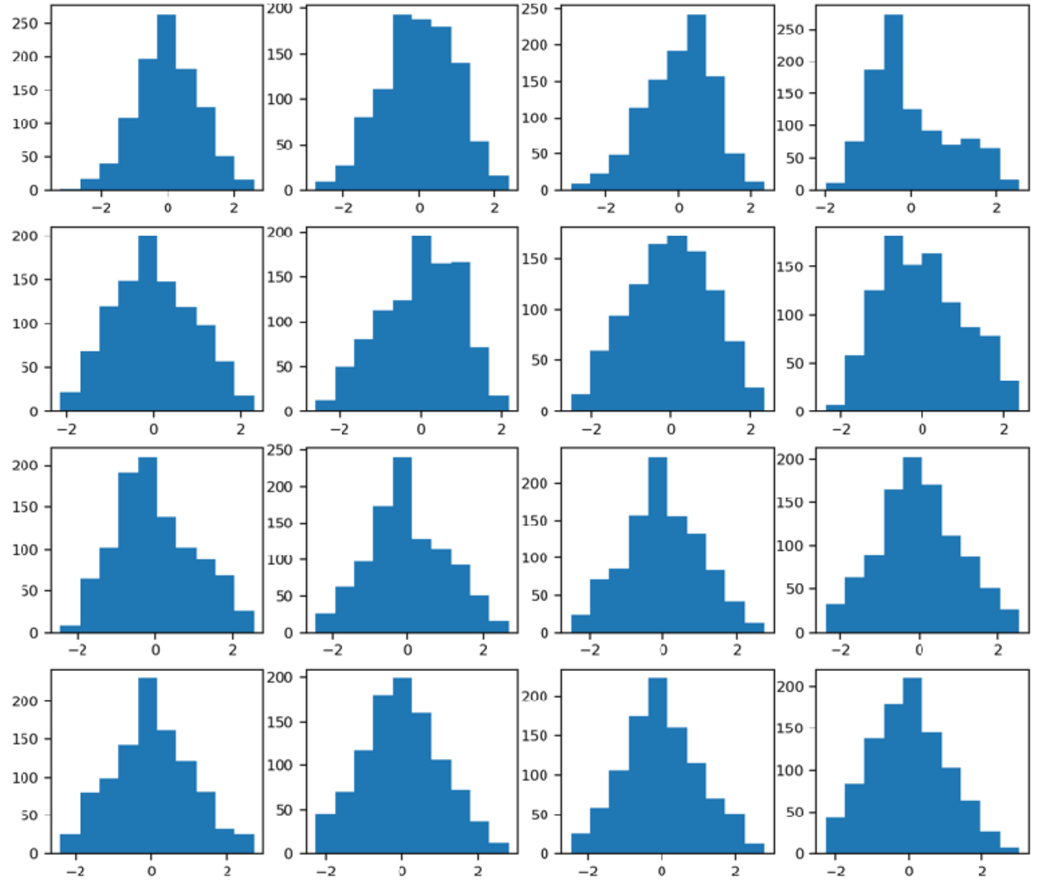
Batch Normalization

-A straightforward way to keep the distribution organized

- 1st layer, 1st epoch; a histogram is from within a mini-batch for a fixed hidden unit output



No BN

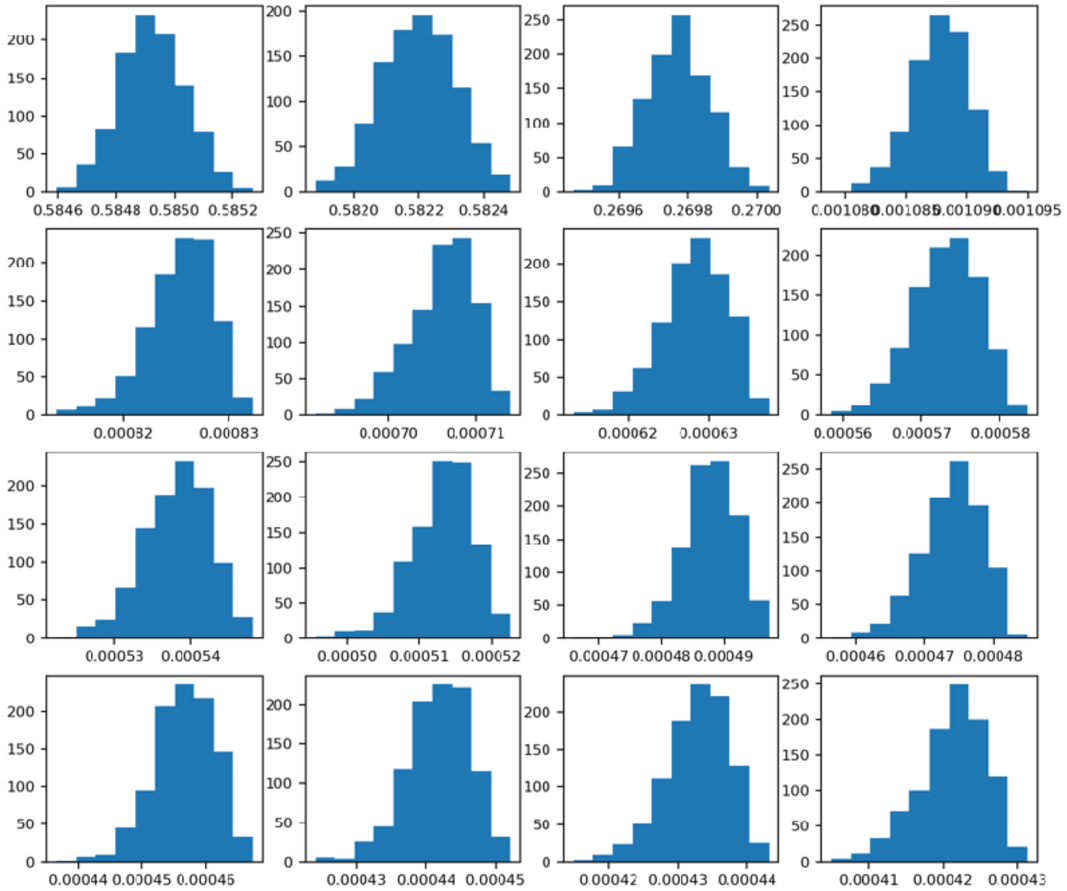


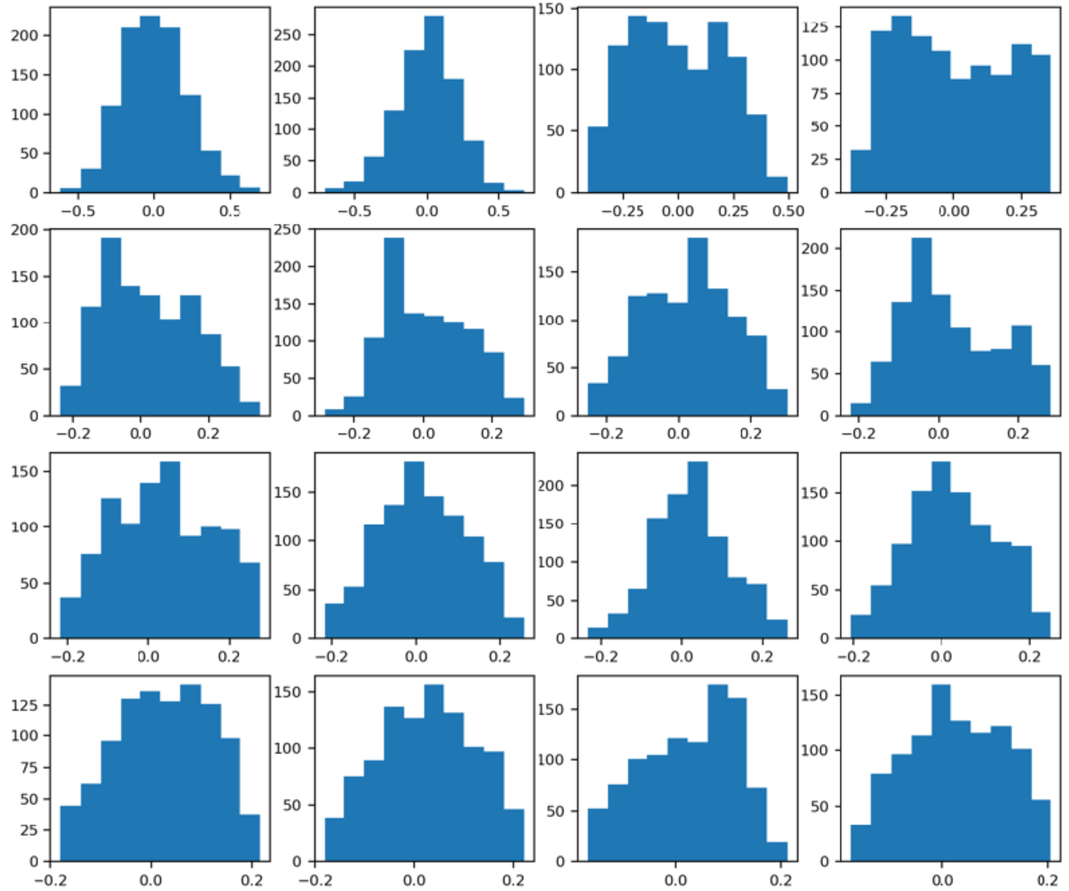
BN

Batch Normalization

-A straightforward way to keep the distribution organized

- 3rd layer, 1st epoch; a histogram is from within a mini-batch for a fixed hidden unit output

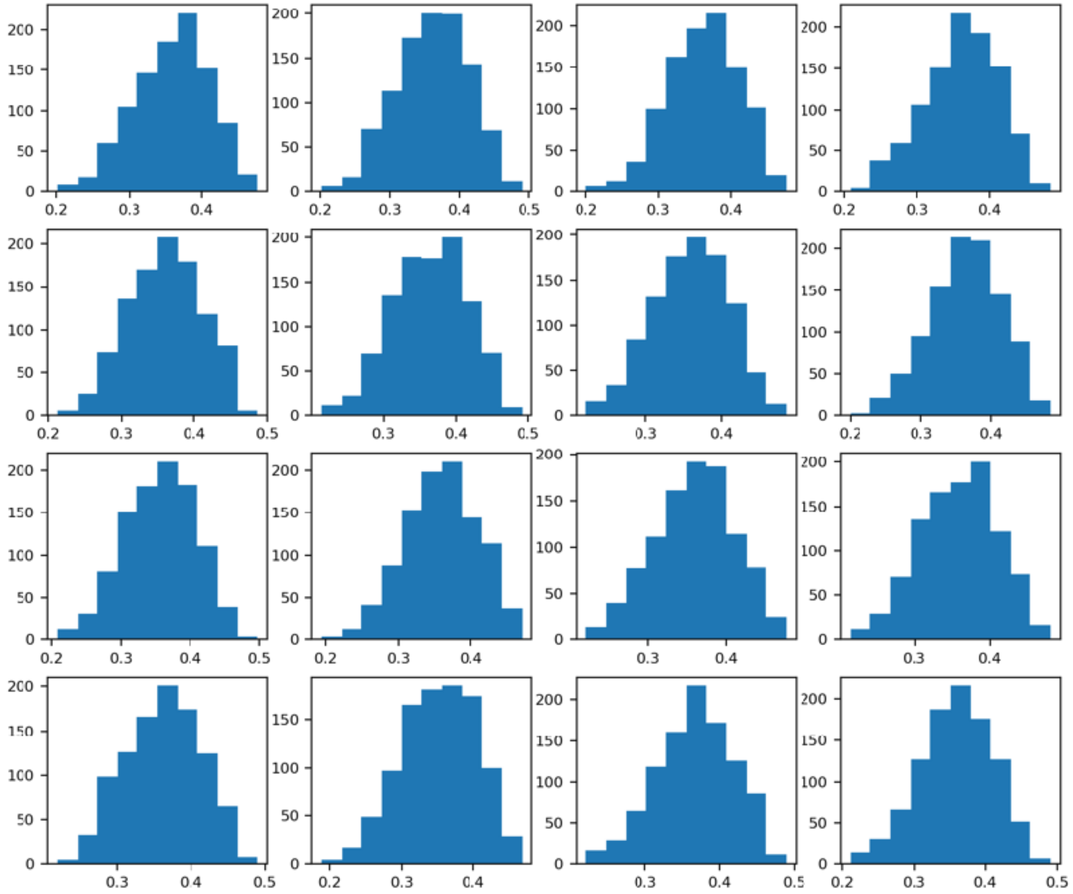


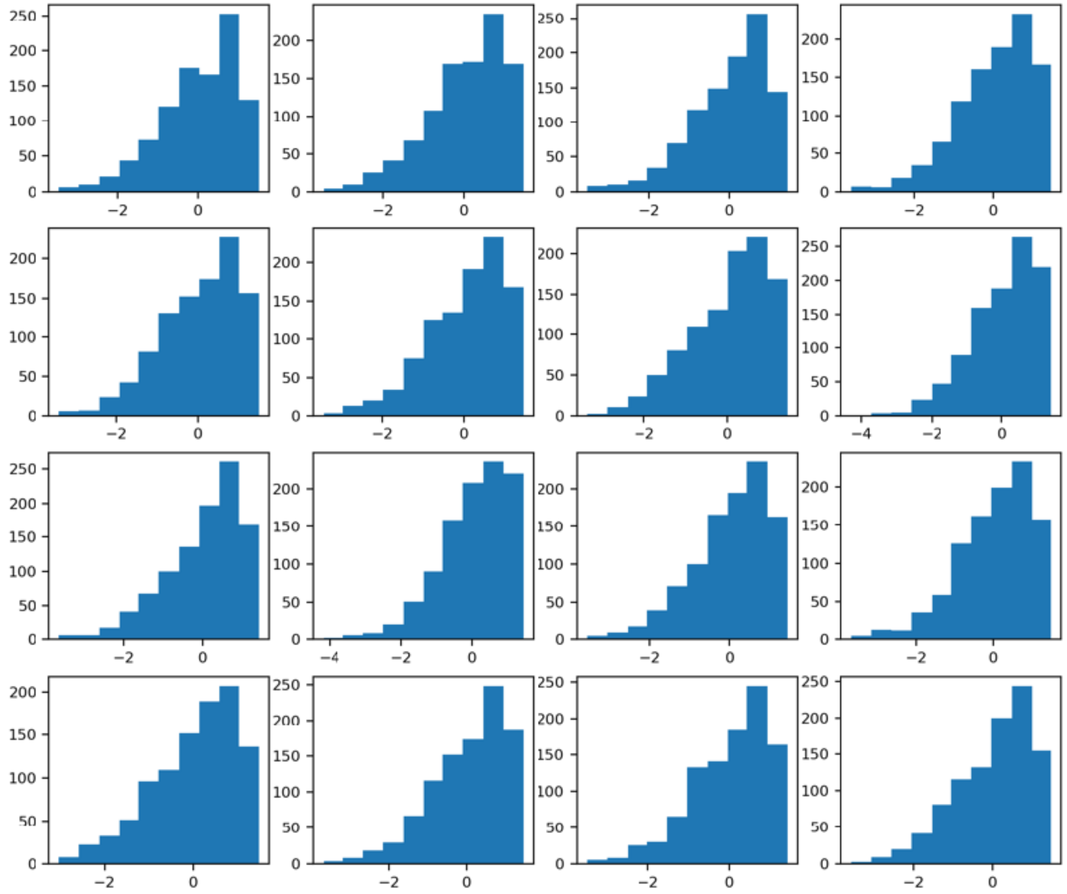


Batch Normalization

-A straightforward way to keep the distribution organized

- 1st layer, 20th epoch; a histogram is from within a mini-batch for a fixed hidden unit output

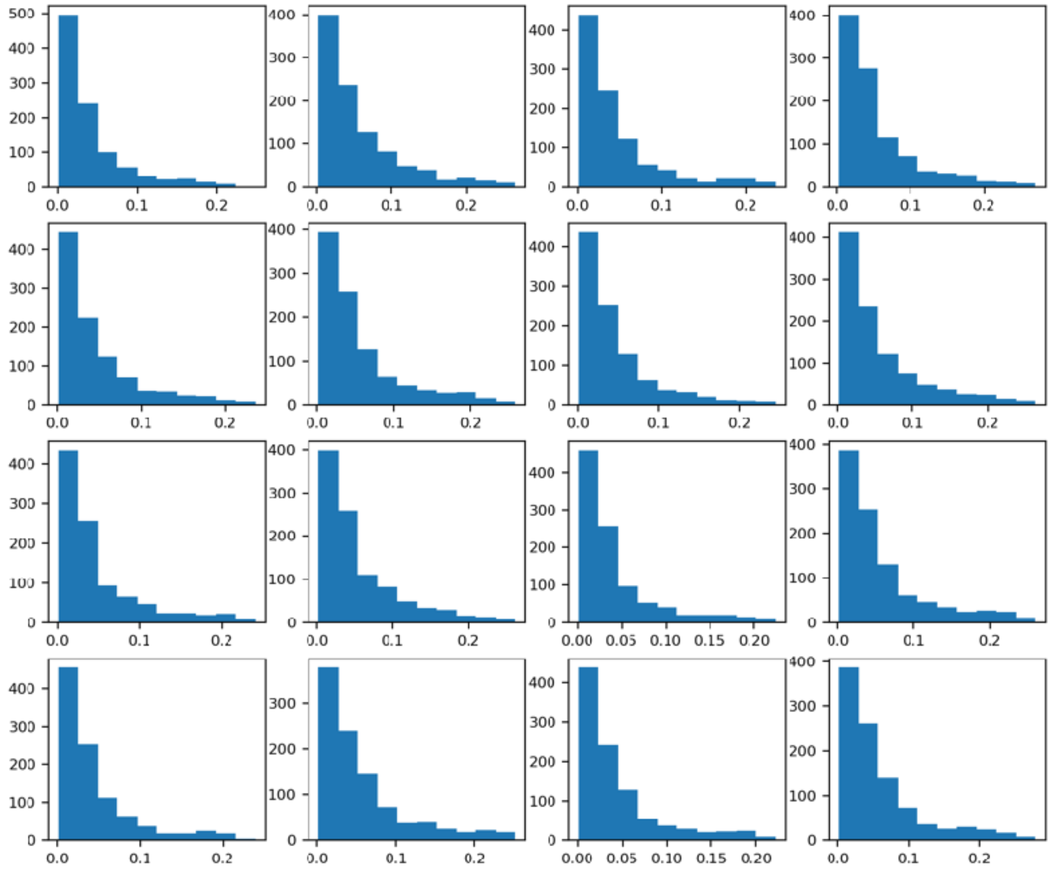


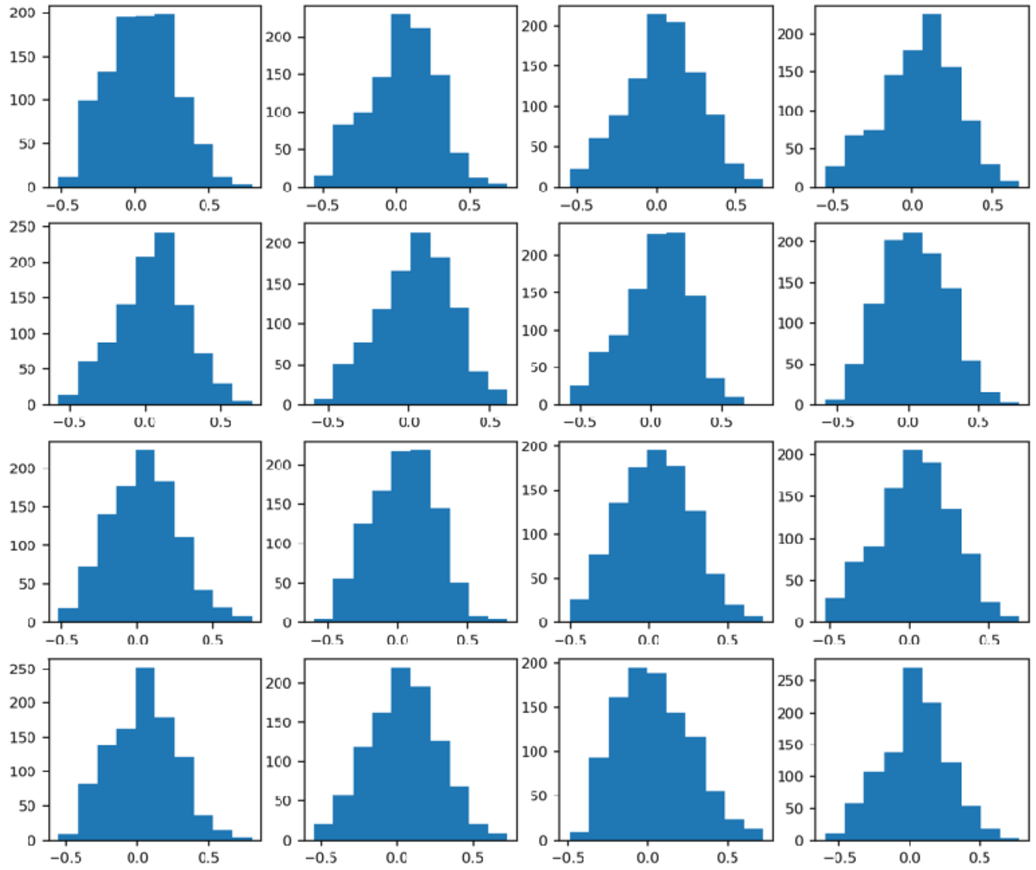


Batch Normalization

-A straightforward way to keep the distribution organized

- 3rd layer, 20th epoch; a histogram is from within a mini-batch for a fixed hidden unit output

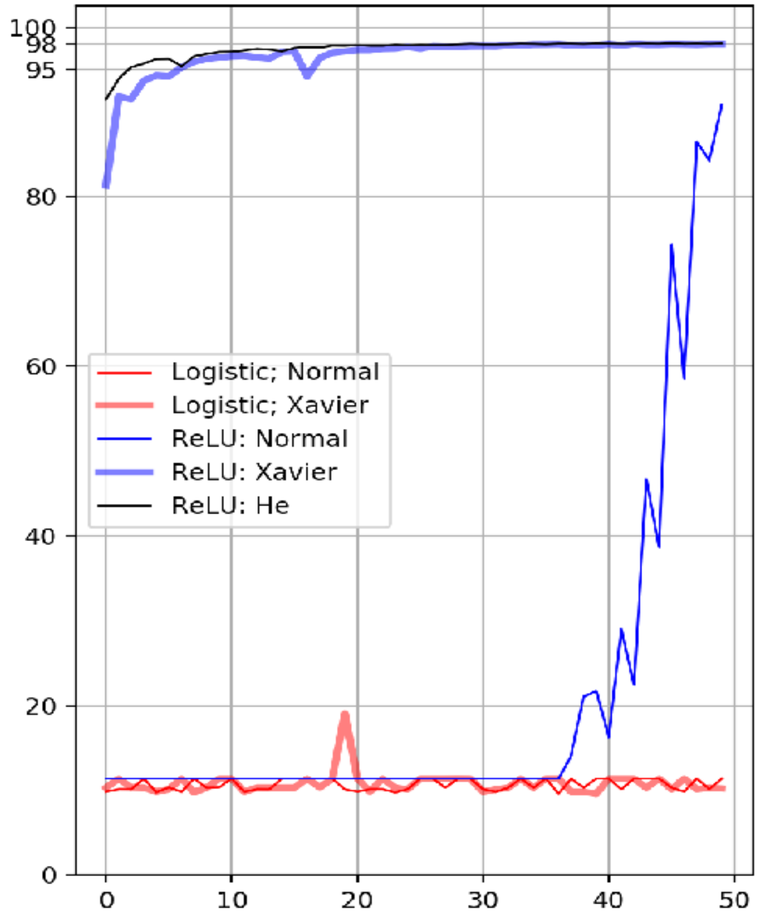




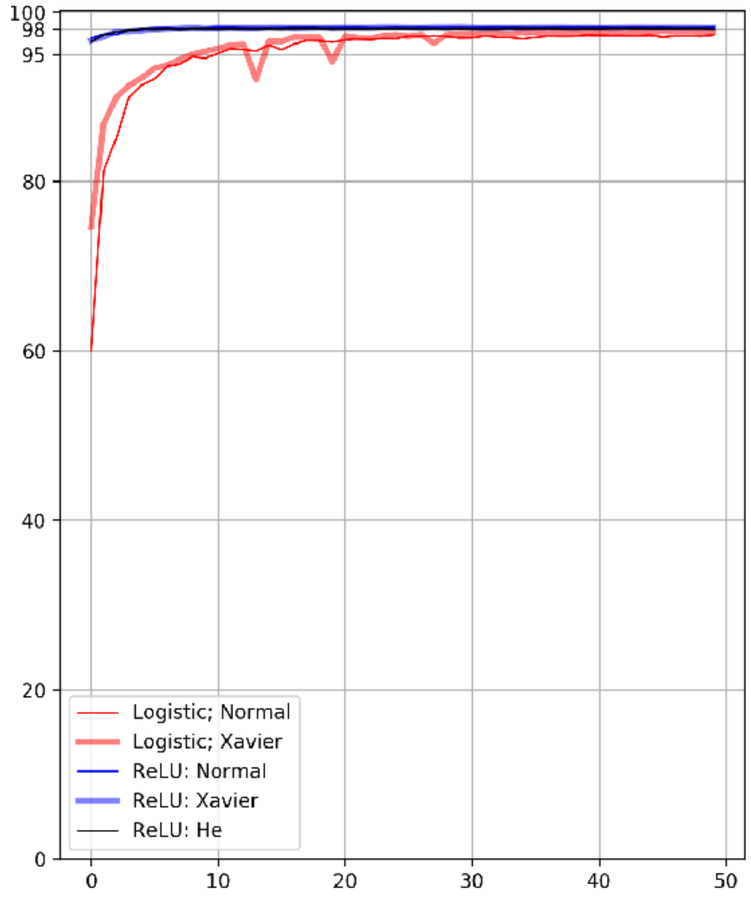
Batch Normalization

-A straightforward way to keep the distribution organized

- Batch normalization reduces the influence of the initialization methods and activation functions



No BN



BN

Momentum and Adaptive Learning Rates

Yes, they are related, too.

Gradient Descent

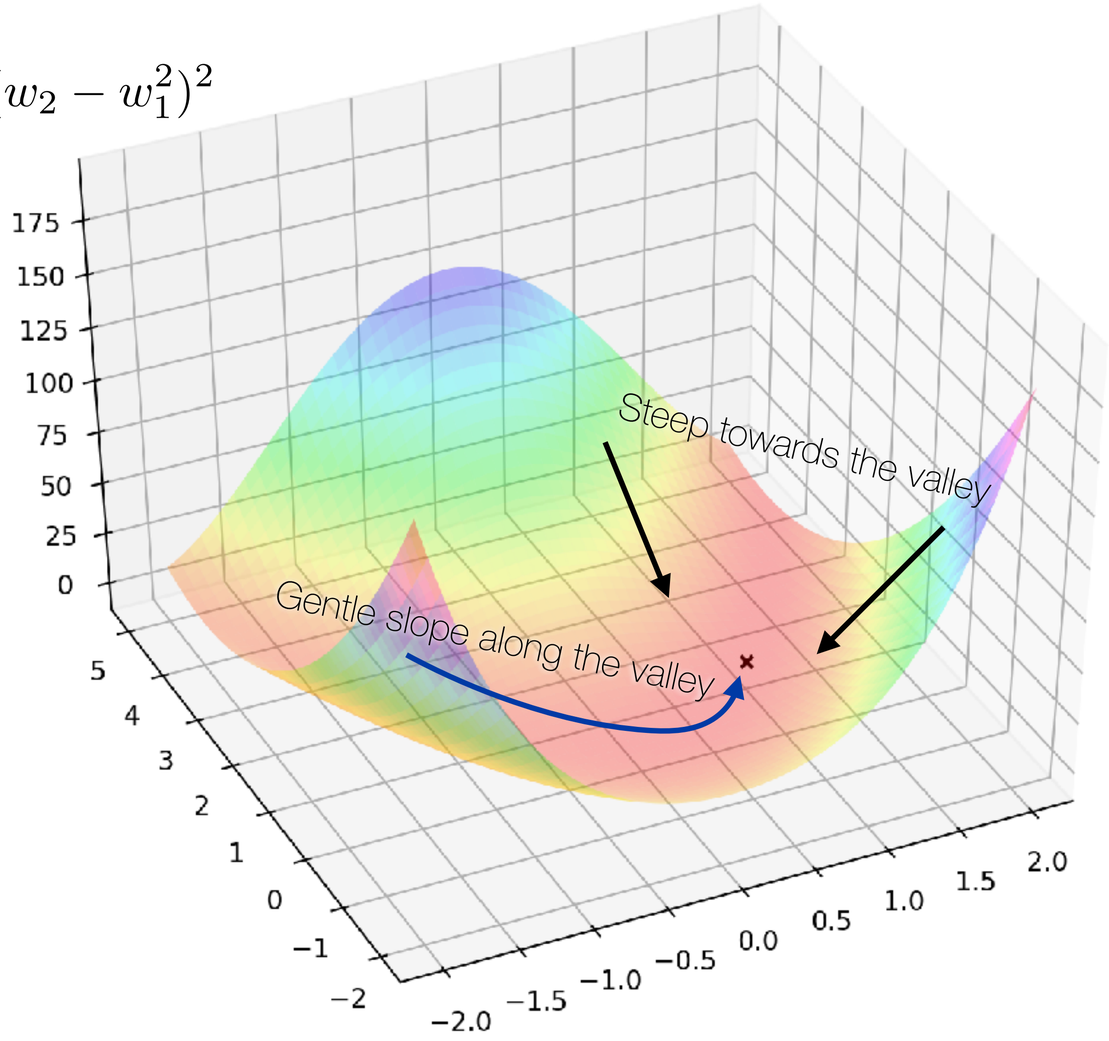
-Aren't you already tired of it?

- The (stochastic) gradient descent method has some problems
  - It assumes the objective function is locally linear
    - Newton's method can fix this, but not suitable for deep learning. Why?
    - Inverse of Hessian matrix is too expensive because of the too many parameters to estimate
    - Quasi-Newton's method is doable, but not so popular choice
  - A fixed learning rate can be either too large or too small
    - Too large LR can overlook an important local minimum
    - Too small LR can stuck in an unimportant local minimum
    - A decaying learning rate can be a reasonable solution, but scheduling is not easy
    - A fixed scalar learning rate might not be best for all parameters
- So far unless I confessed in advance I've used SGD
  - So my previous arguments about initialization and batch normalization can change if I choose another optimization technique

Gradient Descent

-Let's see how bad it can be

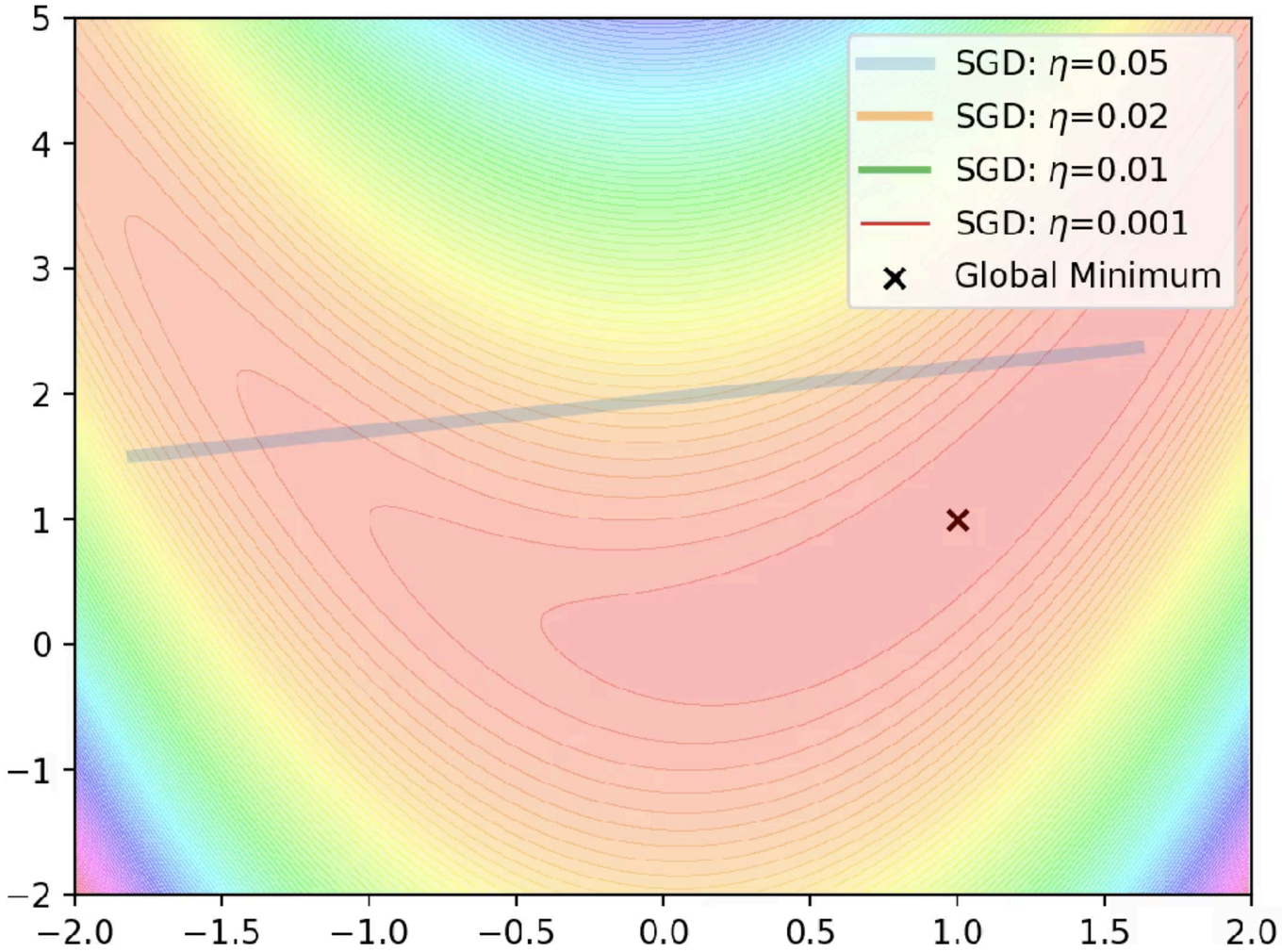
- The Rosenbrock's banana function $f(\boldsymbol{w}) = (x_1 - w_1)^2 + x_2(w_2 - w_1^2)^2$
  - x_1, x_2 : constants (input data)
  - w_1, w_2 : parameters to update
  - $f(\boldsymbol{w})$: the objective function
- The Rosenbrock function is a non-convex function, but it has a global minimum at
  $\boldsymbol{w}^* = [1, 1]^\top \qquad f(\boldsymbol{w}^*) = 0$
  - It's just difficult to find it
- Gradient descent upadates
  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} f(\boldsymbol{x}; \boldsymbol{w})$
- This function is weird
  - It's something like a steep valley
  - where water runs very slow



Gradient Descent

-Let's see how bad it can be

- Choosing the best learning rate is difficult
- The best learning rate is not best everywhere



Momentum

-Let's give it some speed

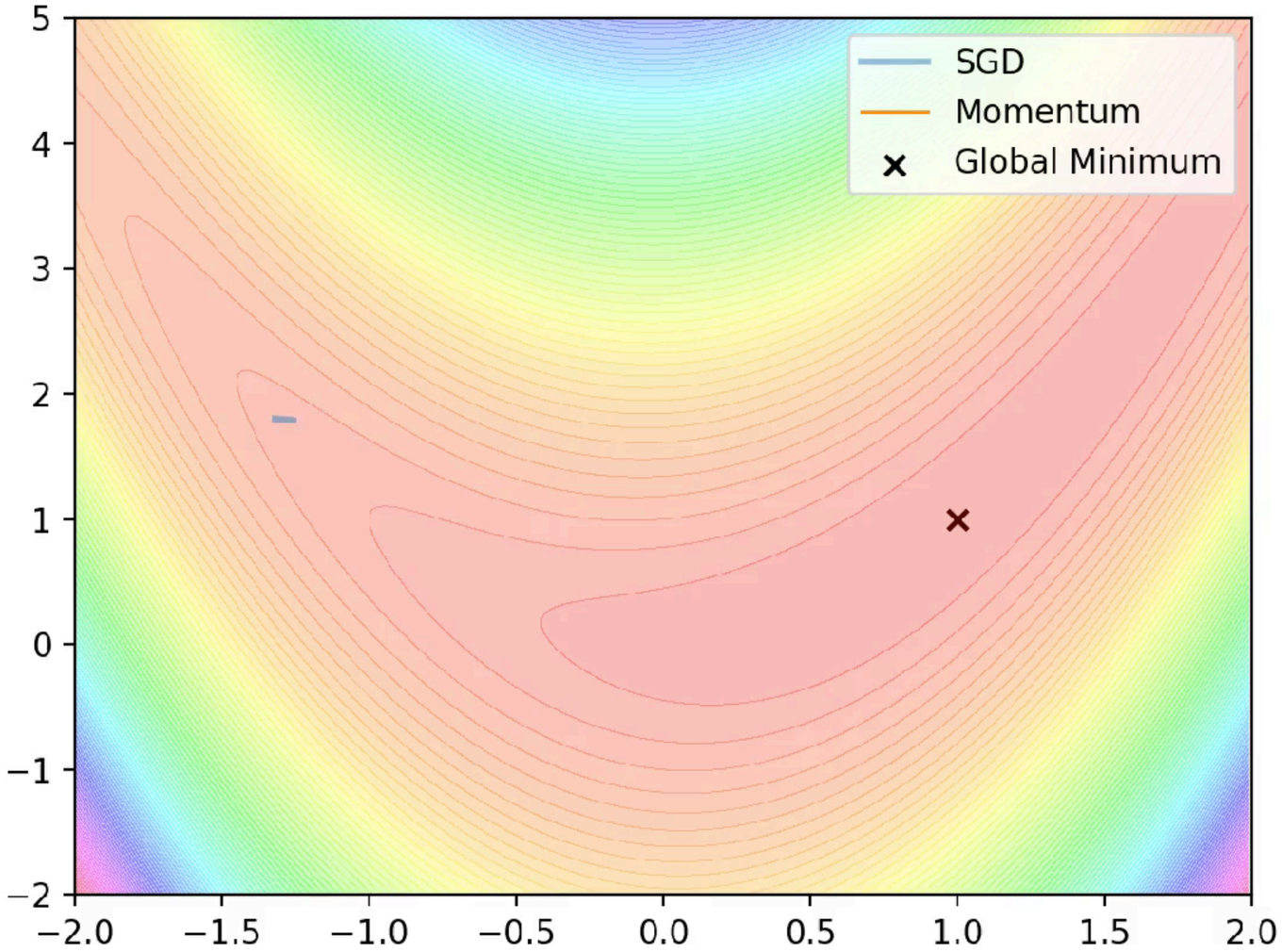
- Velocity

$$\boldsymbol{v} \leftarrow \gamma \boldsymbol{v} - \eta \nabla_{\boldsymbol{w}} f(\boldsymbol{x}; \boldsymbol{w})$$

  - Initially a zero vector,
    but eventually accumulates all the SGD directions
- Update

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{v}$$

- Meaning?
  - If the update has been done
    with a large velocity,
    sudden changes in the SGD directions
    don't affect the update too much
  - Because usually γ is a large number, e.g. 0.9
- See the video for a comparison with SGD

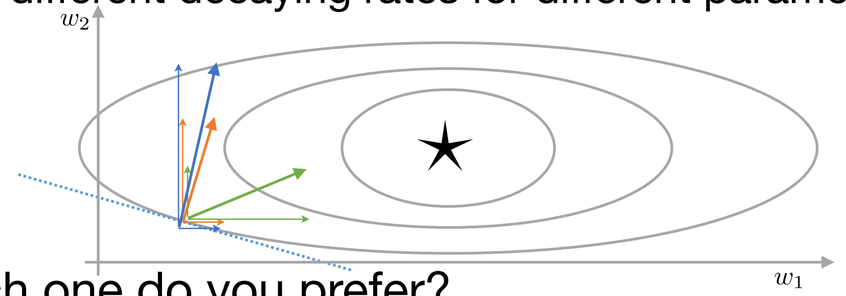


Sutskever, Ilya, et al. "On the importance of initialization and momentum in deep learning." International conference on machine learning. 2013.

AdaGrad

-Per-parameter adaptive learning rate

- It's to gradually decay the learning rate
  - But different decaying rates for different parameters



- Which one do you prefer?

Original GD direction

$$-\nabla_{\boldsymbol{w}} = -\nabla_{w_1} - \nabla_{w_2}$$

Larger LR to the gentle slope

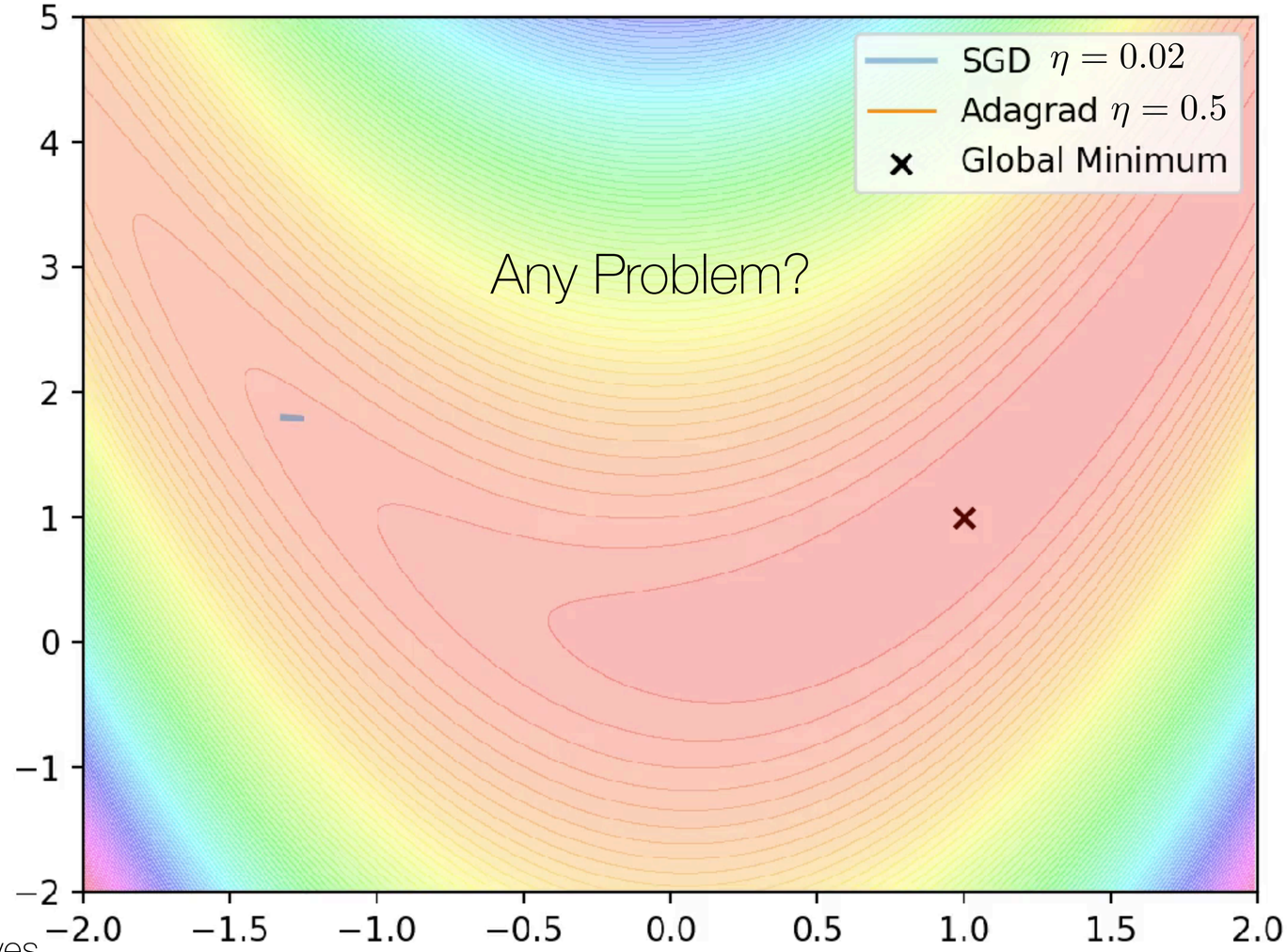
$$-\nabla_{\boldsymbol{w}} = -\eta_1 \nabla_{w_1} - \eta_2 \nabla_{w_2}, \quad \eta_1 > \eta_2$$

Larger LR to the steeper slope

$$-\nabla_{\boldsymbol{w}} = -\eta_1 \nabla_{w_1} - \eta_2 \nabla_{w_2}, \quad \eta_1 < \eta_2$$

- Update
  - Gradient $\boldsymbol{g} \leftarrow \eta \nabla_{\boldsymbol{w}} f(\boldsymbol{x}; \boldsymbol{w})$
  - Accumulated squared gradient $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$
  - Update $\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{\eta}{\epsilon + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$

Large partial derivatives demote LR



Duchi, John, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." Journal of Machine Learning Research 12.Jul (2011): 2121-2159.

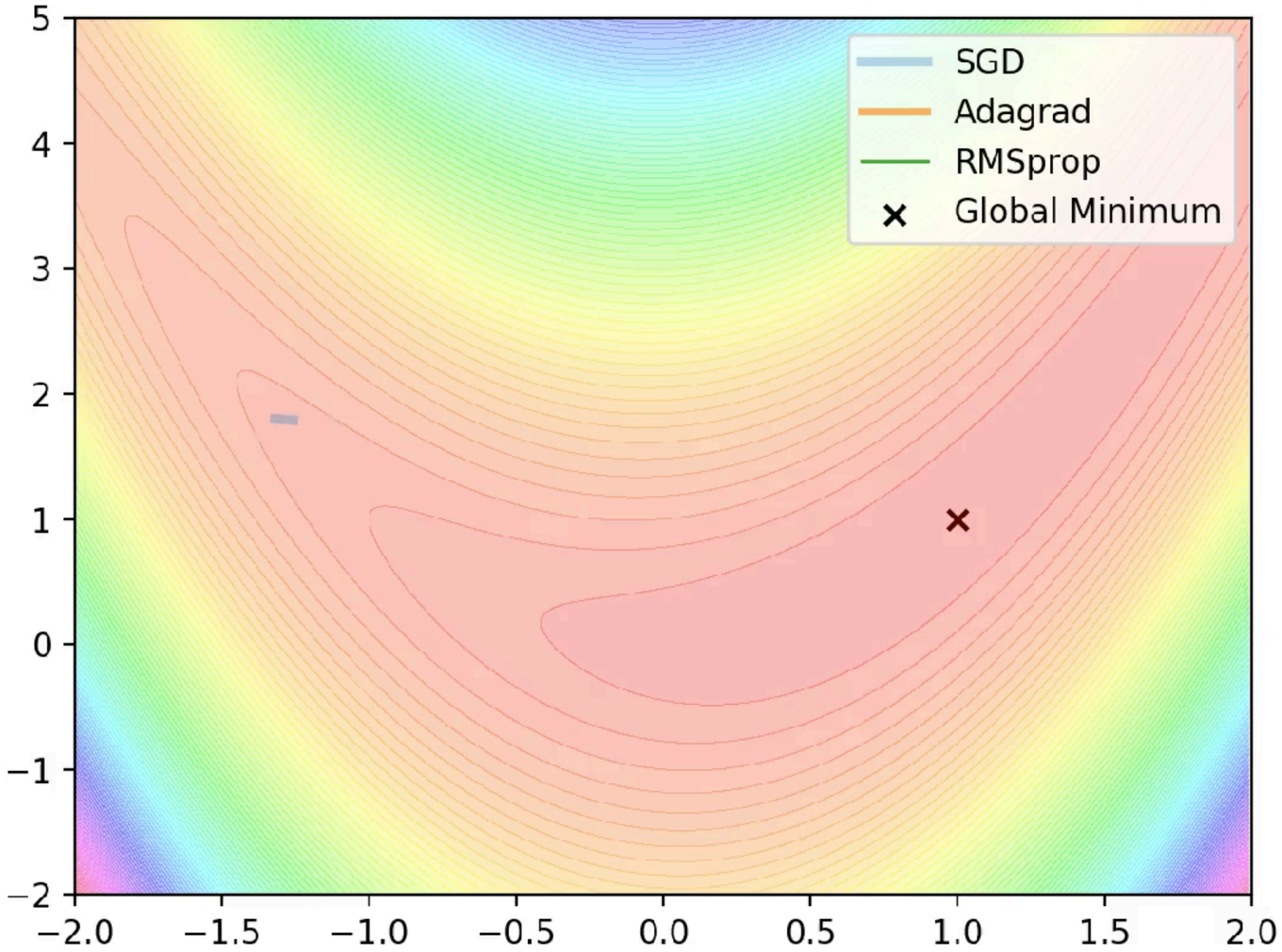
RMSprop

-To improve AdaGrad

- A potential problem with AdaGrad
  - Gradient $\boldsymbol{g} \leftarrow \eta \nabla_{\boldsymbol{w}} f(\boldsymbol{x}; \boldsymbol{w})$
  - Accumulated squared gradient $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$
  - Update $\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{\eta}{\epsilon + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$ — Large partial derivatives demote LR
  - $\boldsymbol{r}$ keeps getting large, and the LR can vanish
- RMSprop
  - $\boldsymbol{r} \leftarrow \gamma \boldsymbol{r} + (1 - \gamma) \boldsymbol{g} \odot \boldsymbol{g}$
  - Meaning?
    - The contribution of the first $\boldsymbol{r}$ exponentially decays over iterations:

      At i-th iteration $\gamma^i \boldsymbol{r}^{(0)}$ $\boldsymbol{g} \odot \boldsymbol{g}$ in the first iteration

    - Prevents $\boldsymbol{r}$ from becoming too large

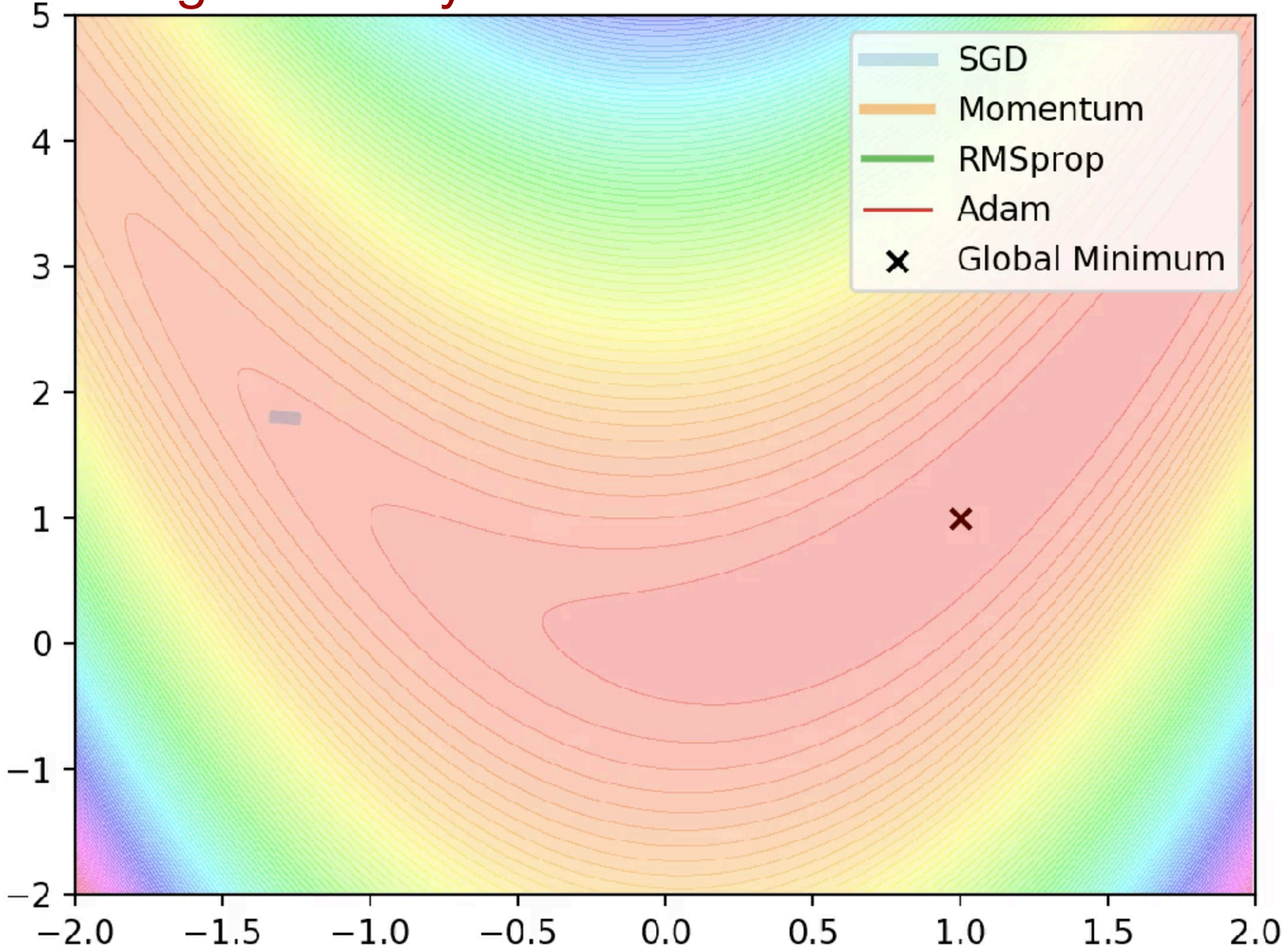


G. Hinton, "Neural Networks for machine learning," Coursera Lecture (2012)

Adam

-Perhaps the most popular choice in deep learning these days

- Combination of RMSprop and momentum
- Momentum (review)
  - Velocity $\boldsymbol{v} \leftarrow \gamma \boldsymbol{v} - \eta \nabla_{\boldsymbol{w}} f(\boldsymbol{x}; \boldsymbol{w})$
  - Update $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{v}$
- RMSprop (review)
  - Gradient $\boldsymbol{g} \leftarrow \eta \nabla_{\boldsymbol{w}} f(\boldsymbol{x}; \boldsymbol{w})$
  - Exponentially accumulated squared gradient $\boldsymbol{r} \leftarrow \gamma \boldsymbol{r} + (1 - \gamma) \boldsymbol{g} \odot \boldsymbol{g}$
  - Update $\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{\eta}{\epsilon + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$
- Adam
  - Gradient $\boldsymbol{g} \leftarrow \eta \nabla_{\boldsymbol{w}} f(\boldsymbol{x}; \boldsymbol{w})$
  - Velocity $\boldsymbol{v} \leftarrow \beta_1 \boldsymbol{v} + (1 - \beta_1) \boldsymbol{g}$ $\quad \hat{\boldsymbol{v}} \leftarrow \frac{\boldsymbol{v}}{1 - \beta_1^i}$
  - Exponentially accumulated squared gradient $\boldsymbol{r} \leftarrow \beta_2 \boldsymbol{r} + (1 - \beta_2) \boldsymbol{g} \odot \boldsymbol{g}$ $\quad \hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \beta_2^i}$
  - Update $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \frac{\hat{\boldsymbol{v}}}{\sqrt{\hat{\boldsymbol{r}}} + \epsilon}$

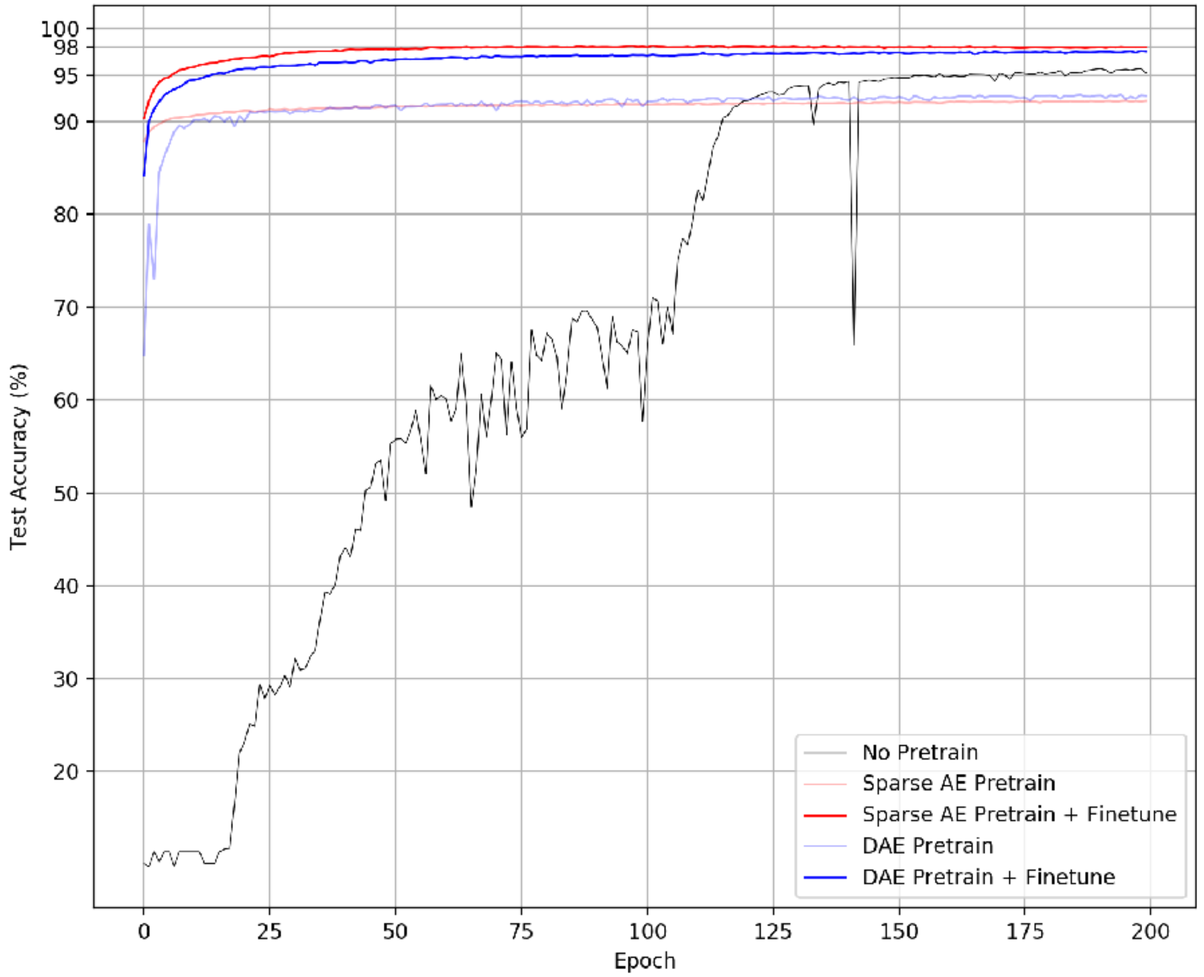


Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

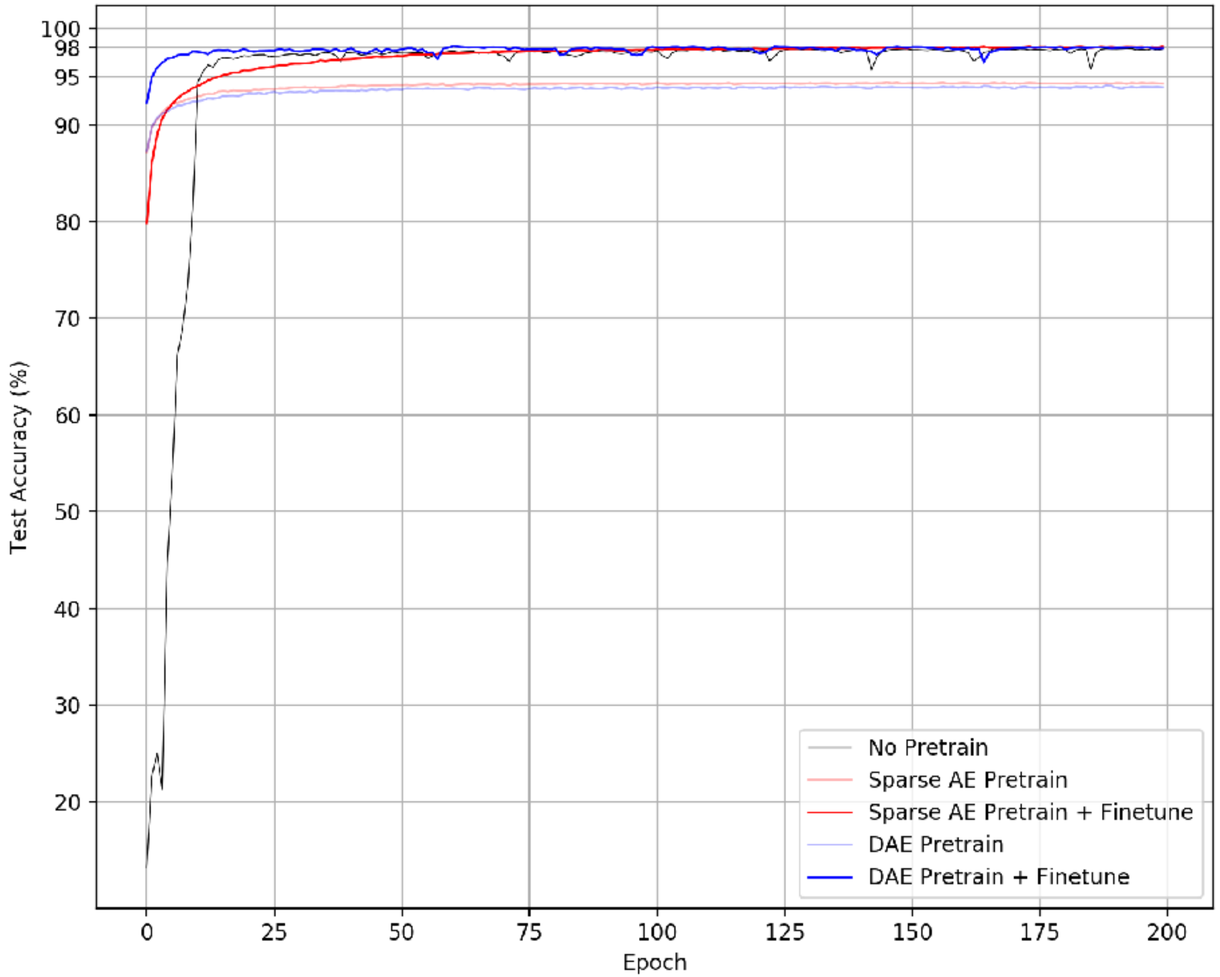
Adam

-On pretrained networks

SGD



Adam

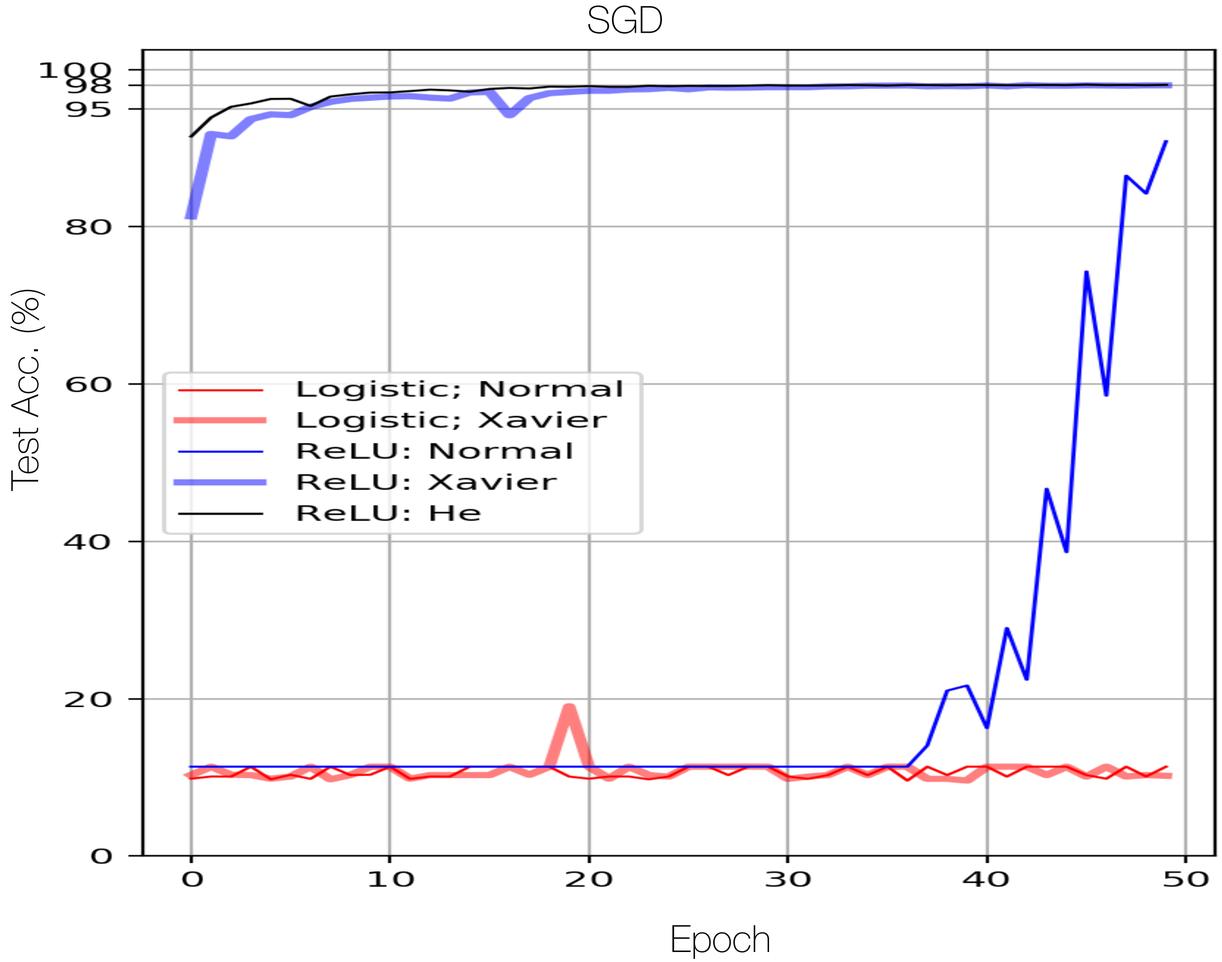


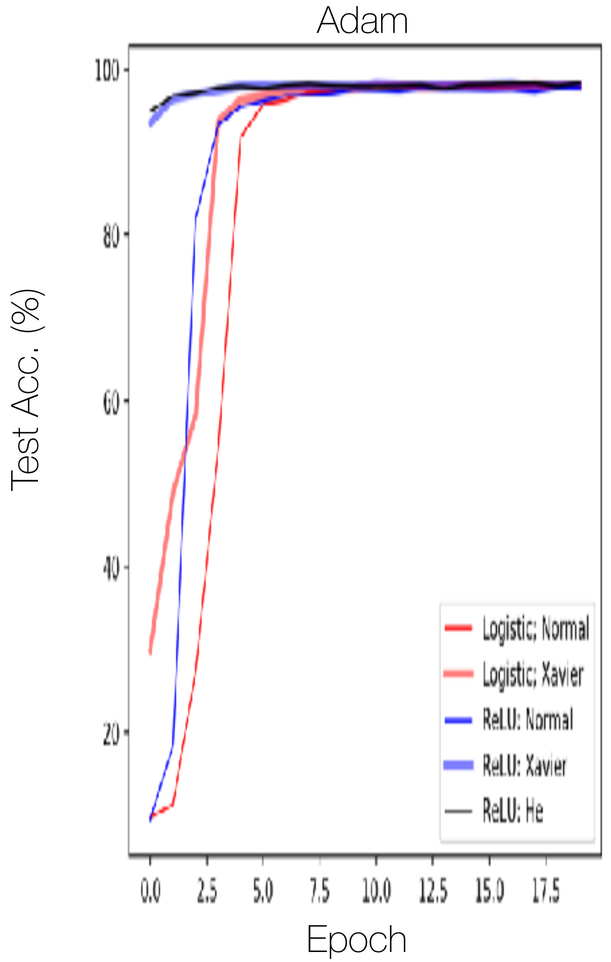
Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

Adam

-On deeper networks with no pretraining

- For a 512X5 network for MNIST



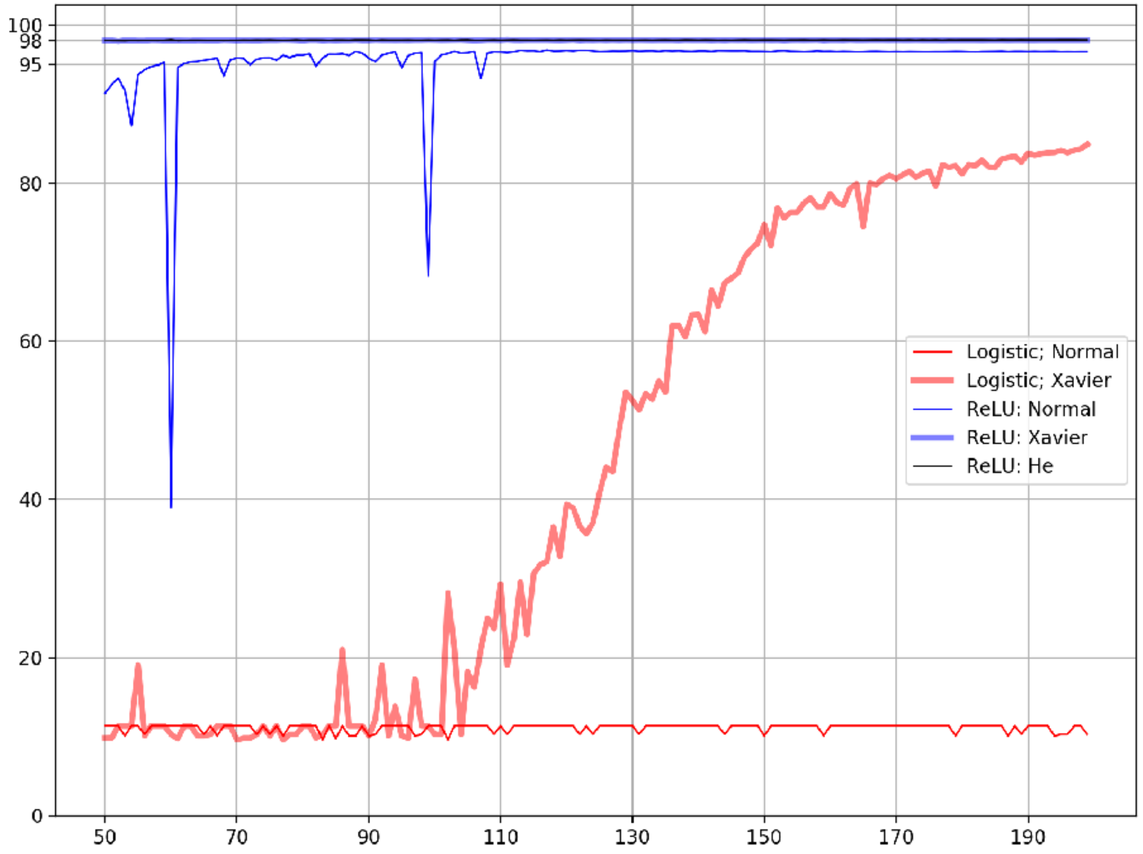


Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

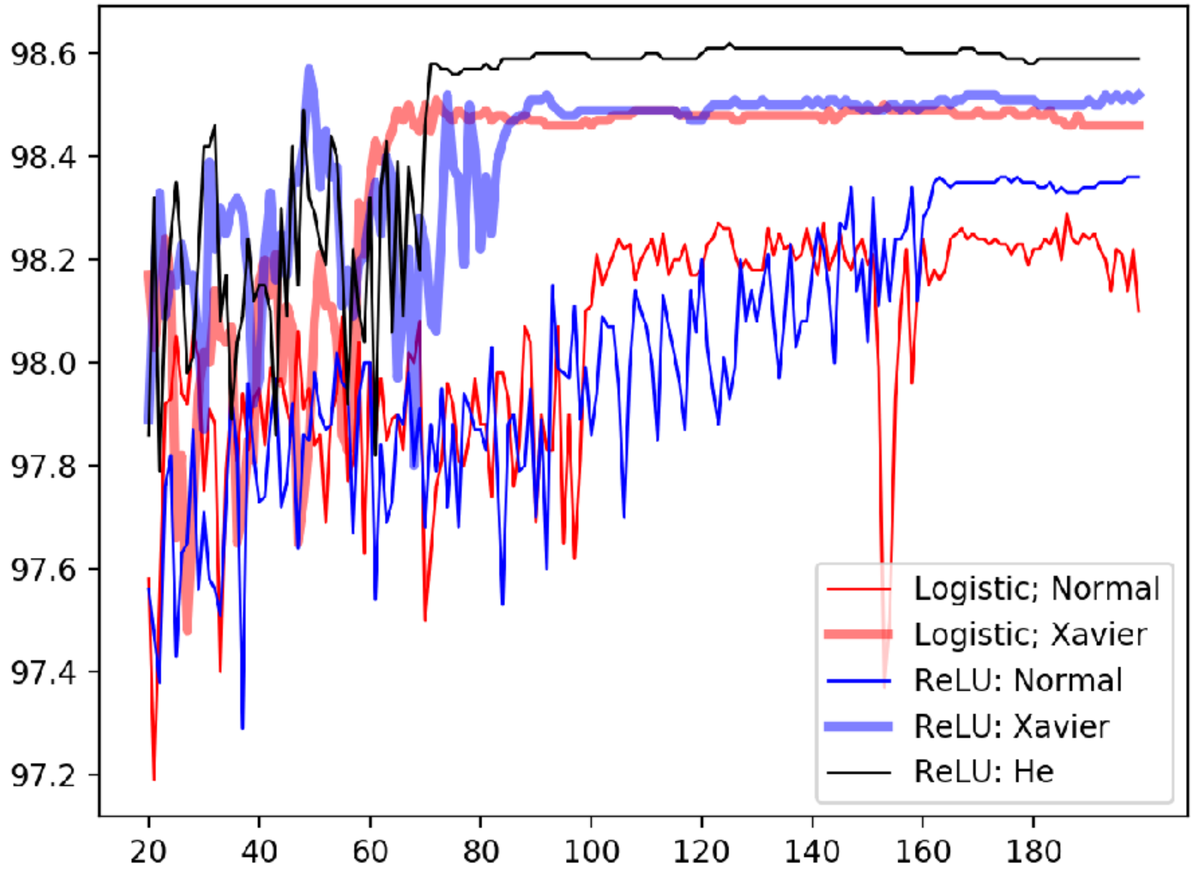
Adam

-On deeper networks with no pretraining

SGD



Adam



Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

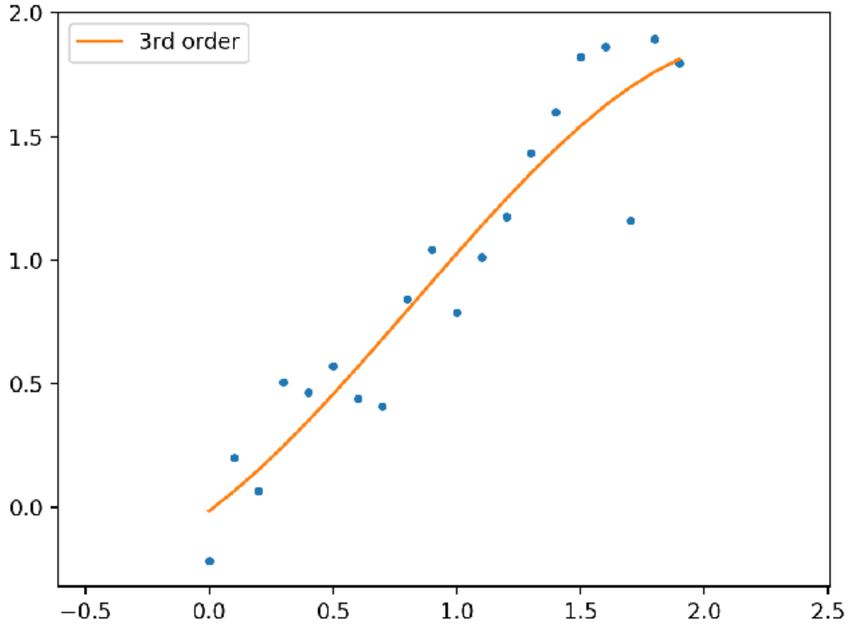
Regularization, Dropout, and Validation

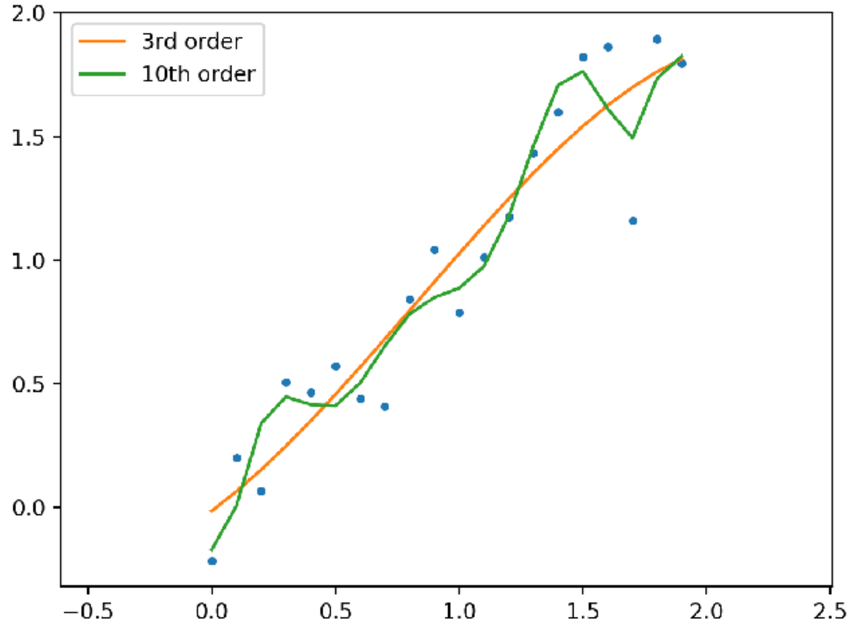
Once again, they are related.

Weight Decaying

-You don't want your weights to be too large

- If your model is reasonably complex (e.g. 3rd order polynomial for a linear dataset)
  - The model tries to fit the data samples by coming up with small weights $[-0.20, 0.50, 0.74, -0.02]$
- If the model is too complex with too many free parameters, then overfitting can happen
  - As a result, the parameters tend to become large $[-25.01, 190.83, -560.18, 727.14, -185.85, -598.32, 741.70, -359.62, 72.98, -2.61, -0.17]$



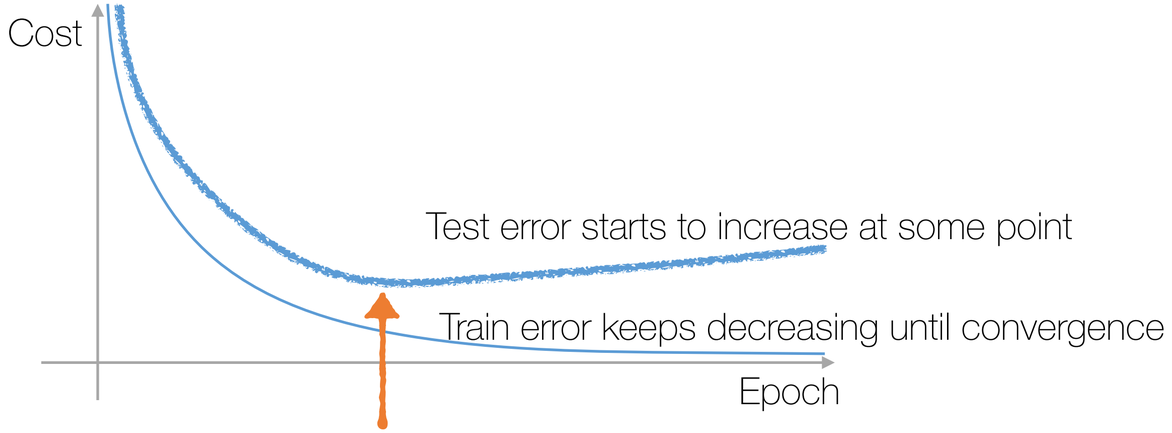


- L2 regularization $\arg\min_{\mathbb{W}} \sum_t \mathcal{D}\left(\boldsymbol{Y}_{:,t} || \mathcal{G}(\boldsymbol{X}_{:,t}; \mathbb{W})\right) + \lambda \sum_{w \in \mathbb{W}} w^2$

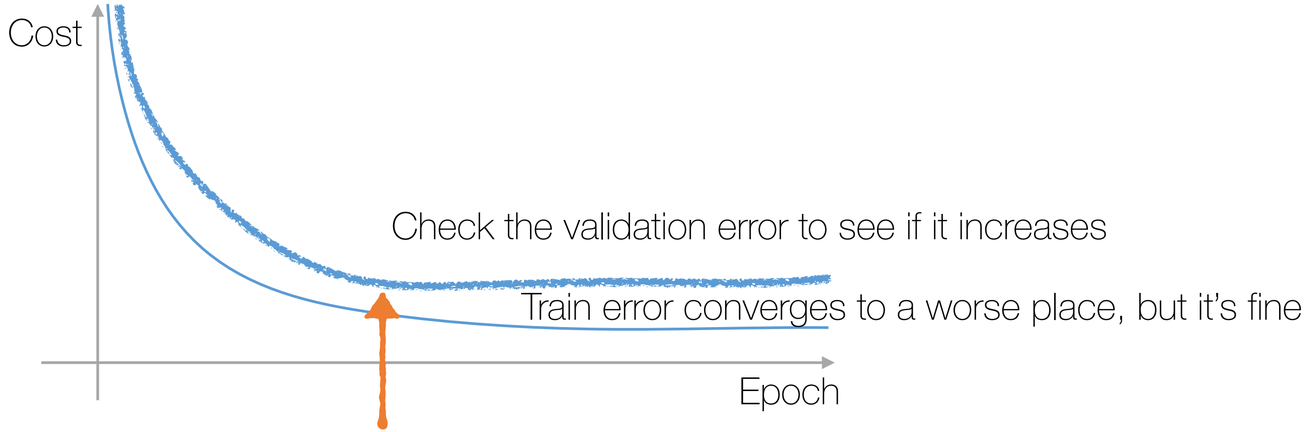
Early Stopping

-Use validation error to prevent overfitting

- So, what happens if overfitting happens?
- What can we do without using the test set?
  - We can set aside some parts of the training set as the validation set
  - Check to see if the validation error increases
- What can we do if the validation error starts to increase?
  - In SGD, we can reduce the learning rate to stay at the local minimum (of the validation error function)
- An indirect way to predict the performance during the test time
- Disadvantage: you can't fully make use of the entire training set
- Cross validation can be also used to find out the optimal hyperparameters
  - e.g. initial LRs, network structures, etc.



We should stop here before the test error goes up

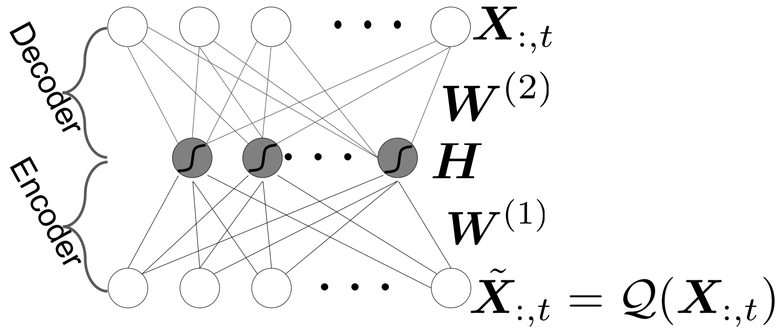


SGD can reduce the learning rate to prevent overfitting

Dropout

-Structural noise injection

- Denoising AutoEncoders
  - Contaminate the input with noise $\tilde{\boldsymbol{X}}_{:,t} = \mathcal{Q}(\boldsymbol{X}_{:,t})$
  - Train the autoencoder to predict the clean input
- e.g. Masking noise $\tilde{\boldsymbol{X}}_{f,t} = \mathrm{b}\boldsymbol{X}_{f,t}, \quad b \sim \mathrm{Bern}(p)$
  - At each iteration you start from a corrupted version of the dataset
  - Still has to do your best to minimize the error
  - This can be seen as noise injection to prevent overfitting, too



Vincent, Pascal, et al. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." Journal of Machine Learning Research 11.Dec (2010): 3371-3408.

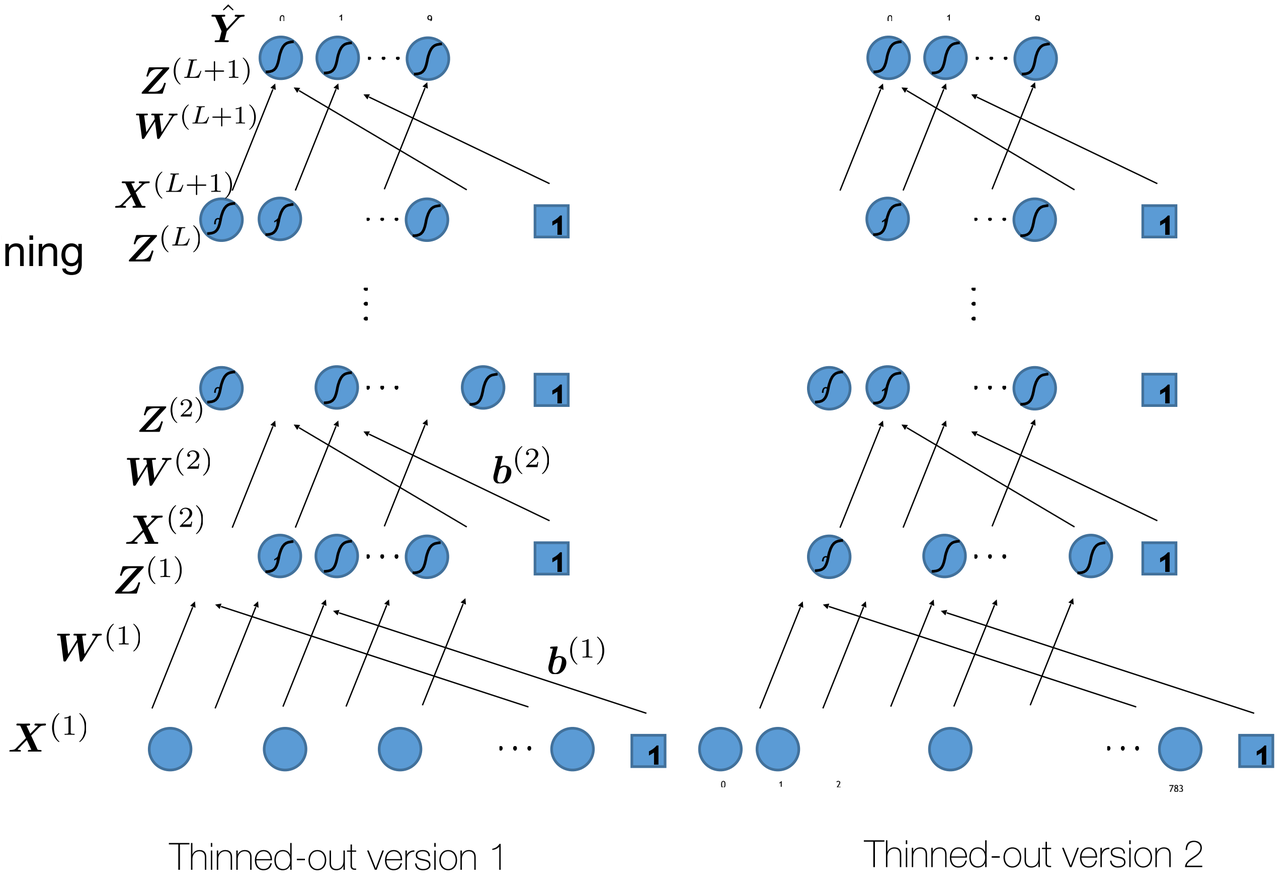
Dropout

-Structural noise injection

- For each data sample, and its feedforward
  - Dropout randomly turns off the input units to the layer

$$\boldsymbol{Z}_{i,t}^{(l)} \leftarrow \sum_j \boldsymbol{W}_{i,j}^{(l)} \boldsymbol{M}_{j,t} \boldsymbol{X}_{j,t}^{(l)} + \boldsymbol{b}_i^{(l)} \qquad \boldsymbol{M}_{j,t} \sim \text{Bern}(p)$$

- So, the masking noise is injected to every layer
  - Kinda similar to stacked DAE
  - But, this is done during the full network training, not pretraining
  - Network is robust to the masking noise
- Network is robust to the structural changes
  - Every feedforward done in a thinned-out network
- Ensemble
  - Each data sample predicts the output using a subsampled version of the network
  - Can be seen as an ensemble technique
- Testing time $\boldsymbol{Z}_{i,t}^{(l)} \leftarrow \sum_j p\boldsymbol{W}_{i,j}^{(l)} \boldsymbol{X}_{j,t}^{(l)} + \boldsymbol{b}_i^{(l)}$
  - Because the weights are trained to handle less input units



Thinned-out version 1

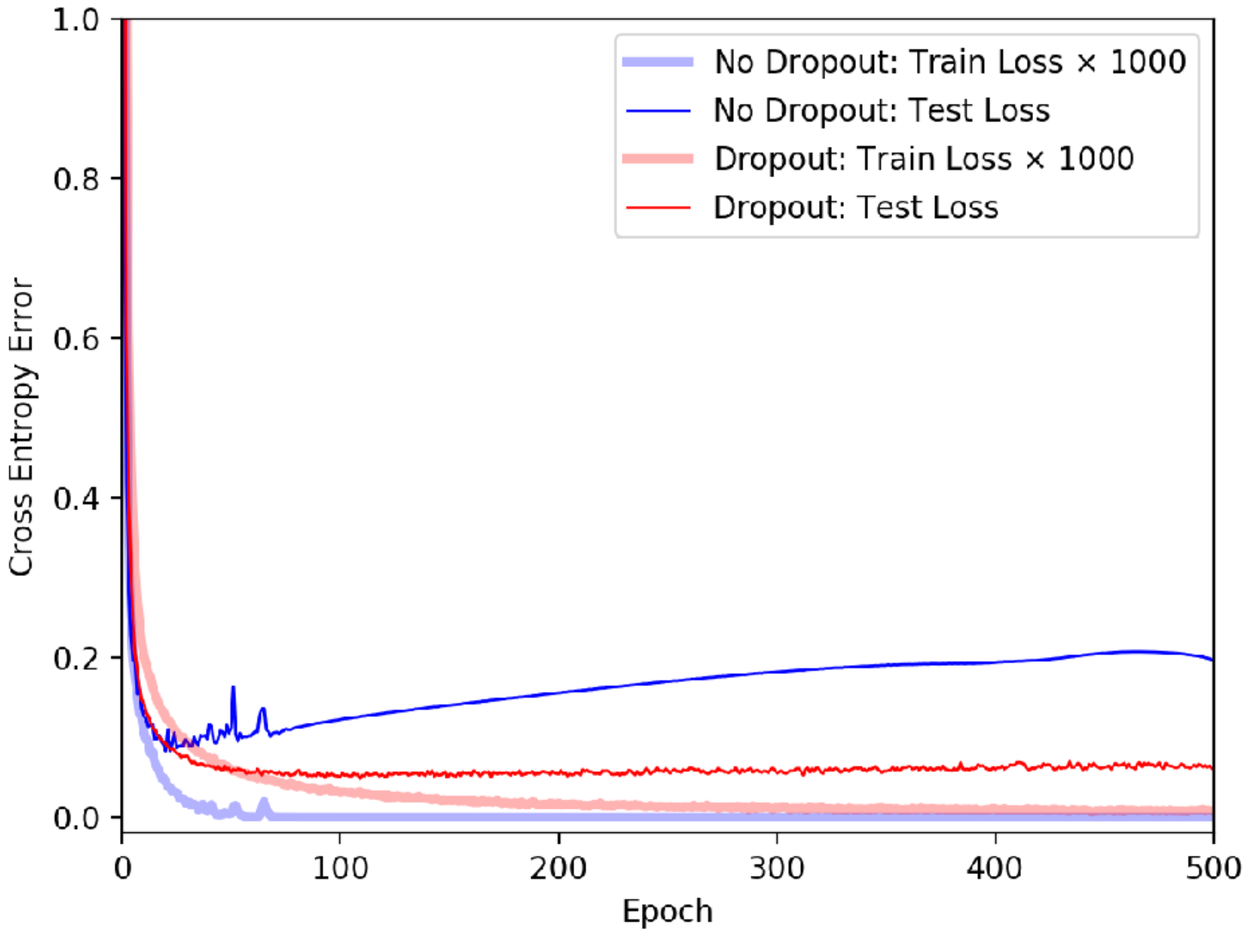
Thinned-out version 2

Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research 15.1 (2014): 1929-1958.

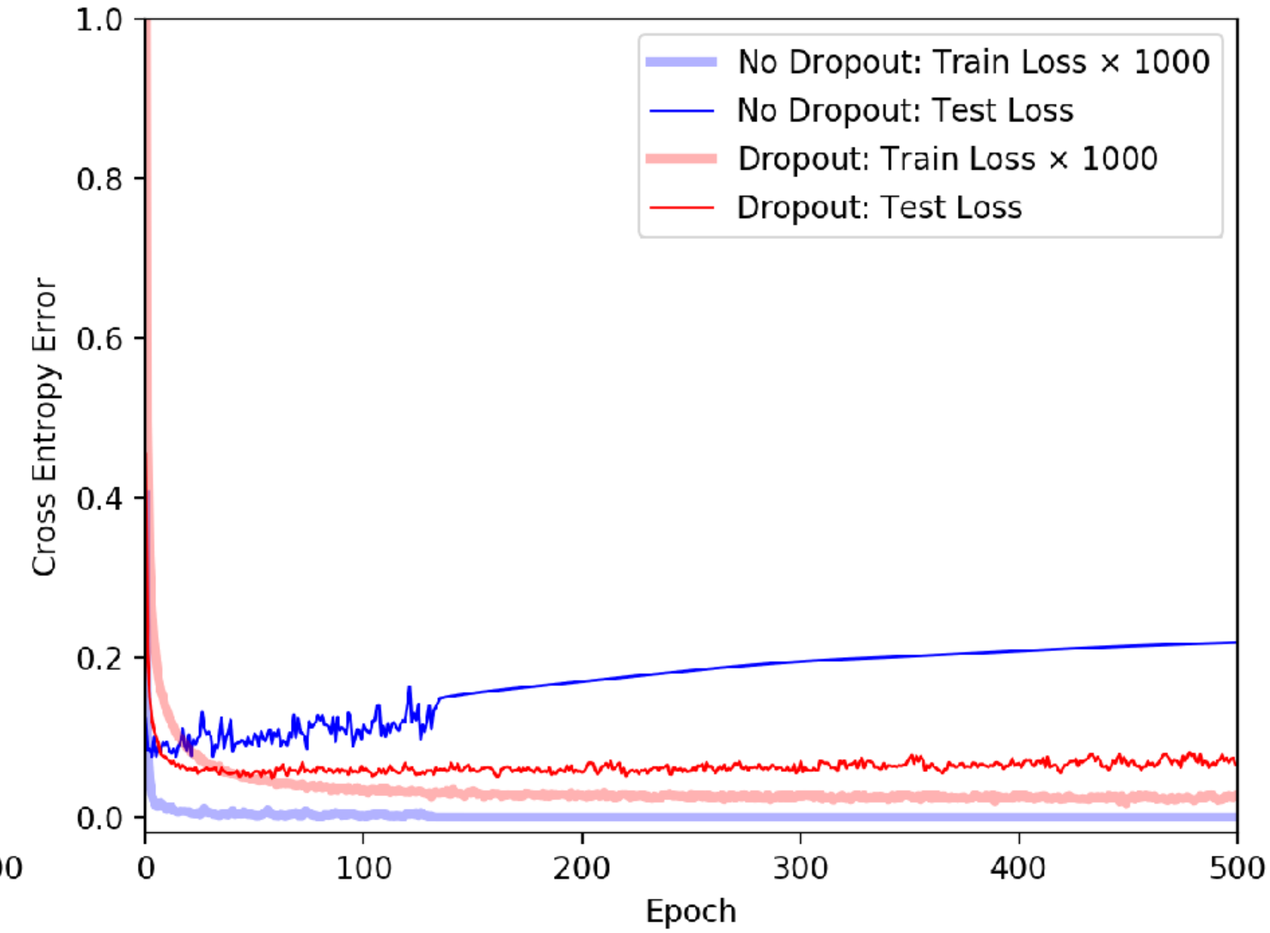
Dropout

-Structural noise injection

1024X5 Logistic



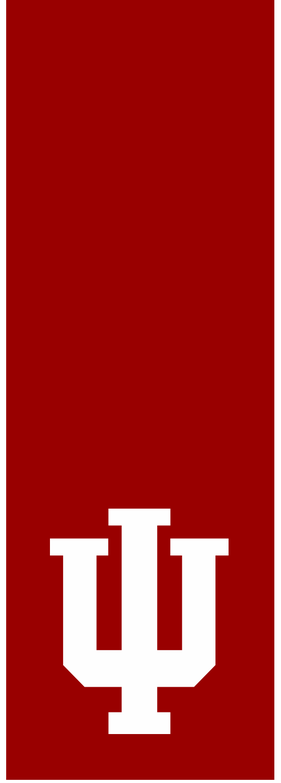
1024X5 ReLU



Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research 15.1 (2014): 1929-1958.

Reading

- All the paper cited
- Textbook Chapter 7, 8



Thank You!