11-695: AI Engineering
Other Techniques II

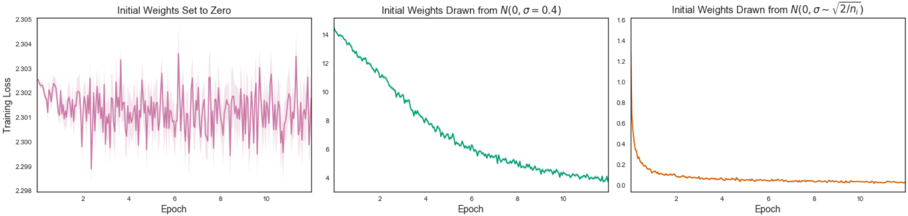
LTI/SCS

Spring 2020

Outline

1. Initialization
2. Weights regularization
3. Dropout

Initialization of Weights and Biases

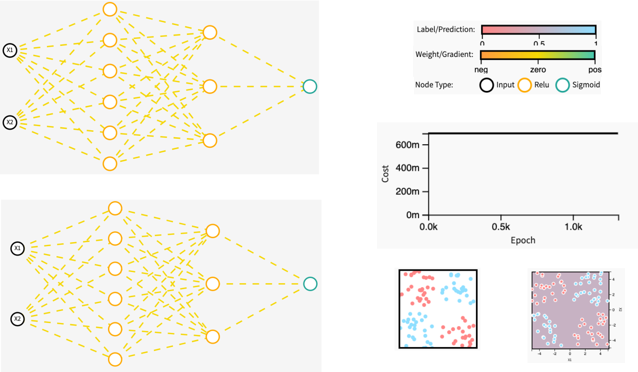


- Training NNs is difficult![1]
- Weight init has an important role in training NNs

[1] http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

Image Credit: Andre Perunicic

Zero Initialization

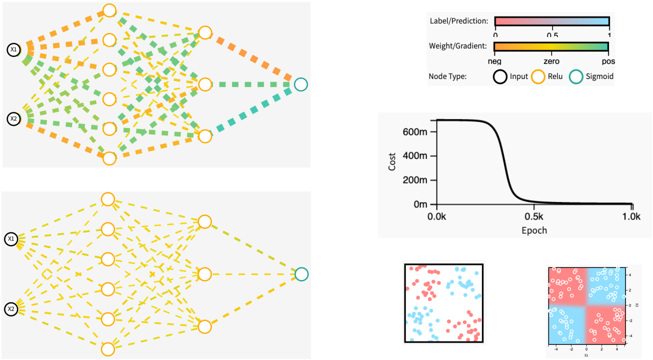


- A big NO, would create a symmetry between hidden layers[2]
- Bias can be initialized zeros, because weights take care of avoiding symmetry

[2] https://arxiv.org/pdf/1206.5533.pdf

Image Credit: https://www.deeplearning.ai/ai-notes/initialization/

Too small Initialization

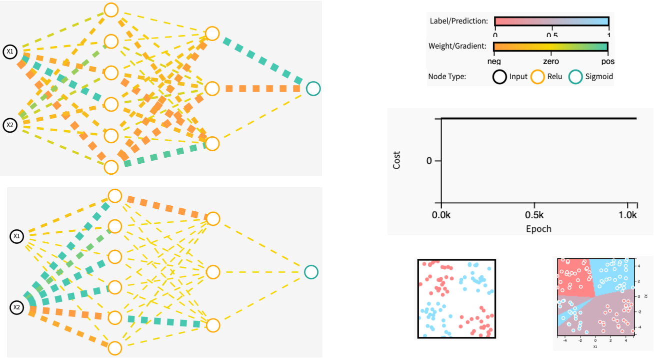


- Gradient vanishing problem

Image Credit: `https://www.deeplearning.ai/ai-notes/initialization/`

Too large Initialization

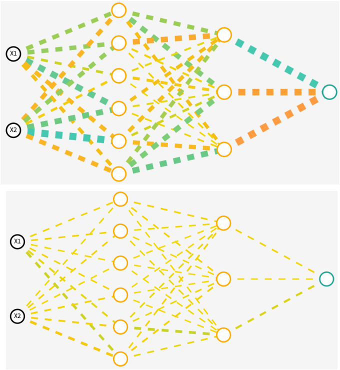
Carnegie Mellon

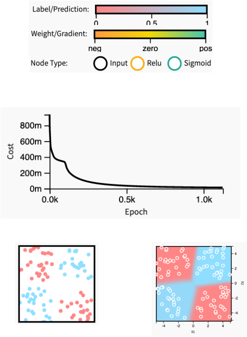


- Gradient exploding problem

Image Credit: `https://www.deeplearning.ai/ai-notes/initialization/`

Proper Initialization

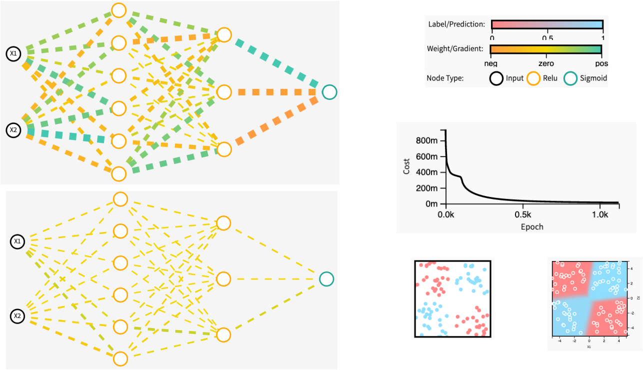




- Would have no such problems

Image Credit: `https://www.deeplearning.ai/ai-notes/initialization/`

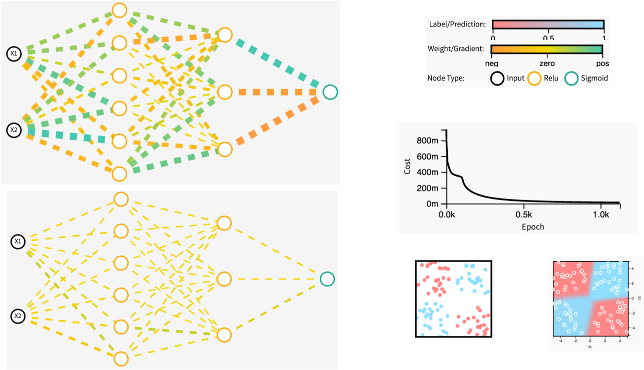
Proper Initialization



- Would have no such problems
- But what it means by being “proper”

Image Credit: `https://www.deeplearning.ai/ai-notes/initialization/`

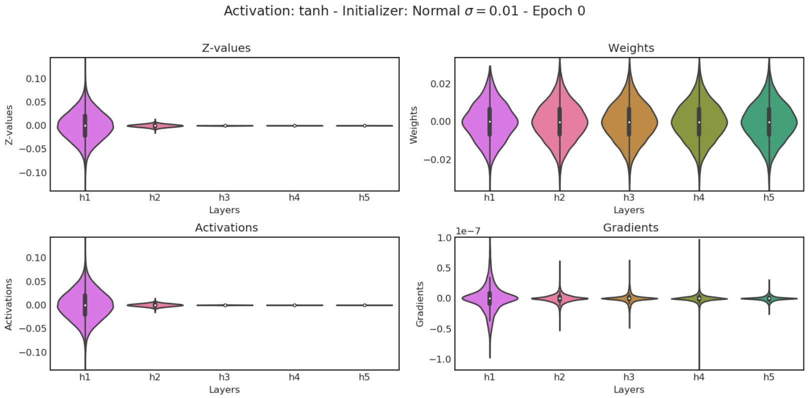
Proper Initialization



- Would have no such problems
- But what it means by being “proper”
  - Mean should center around zero
  - Variance should not shift between layers

Image Credit: https://www.deeplearning.ai/ai-notes/initialization/

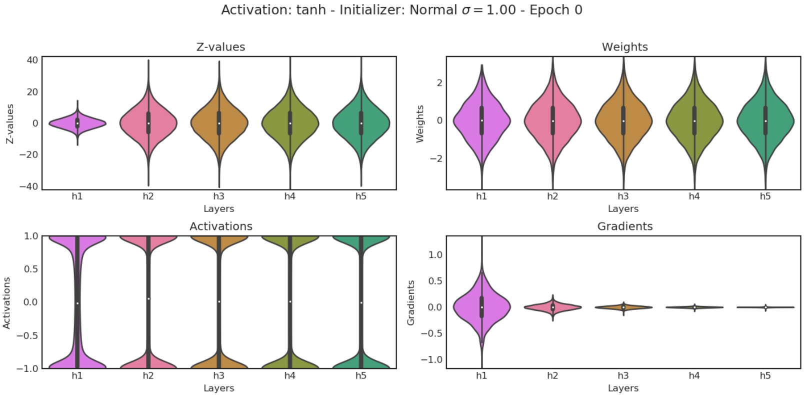
Random Initialization



- Normally from $\mathbf{N}(\mathbf{0}, \sigma^2)$
- Variance *matters*

Image Credit: Daniel Godoy

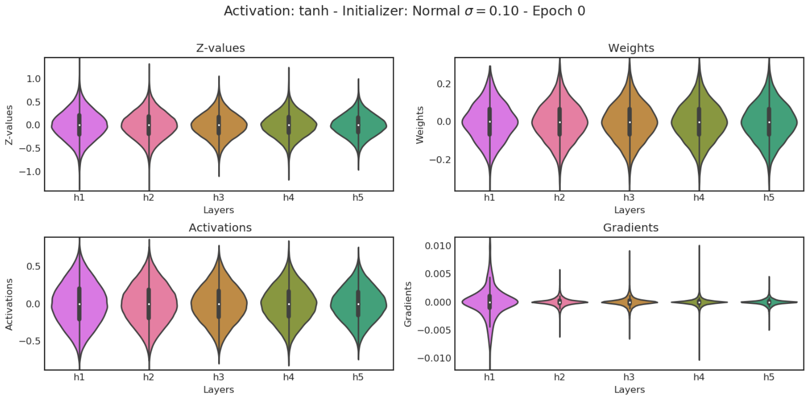
Random Initialization



- Normally from $\mathbf{N}(\mathbf{0}, \sigma^2)$
- Variance *matters*

Image Credit: Daniel Godoy

Random Initialization



- Normally from $\mathcal{N}(\mathbf{0}, \sigma^2)$
- Variance *matters*

Image Credit: Daniel Godoy

(Xavier) Glorot Initialization[3]

Carnegie Mellon

- Forward

$$\sigma(W) = \sqrt{\frac{1}{\text{fan\_in}}} \quad \text{(normal)} \qquad \sigma(W) = \sqrt{\frac{3}{\text{fan\_in}}} \quad \text{(uniform)}$$

- Backward

$$\sigma(W) = \sqrt{\frac{1}{\text{fan\_out}}} \quad \text{(normal)} \qquad \sigma(W) = \sqrt{\frac{3}{\text{fan\_out}}} \quad \text{(uniform)}$$

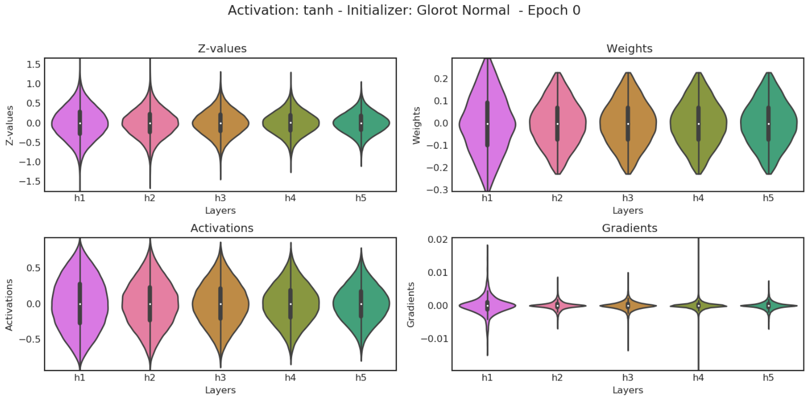
- Combined:

$$\sigma(W) = \sqrt{\frac{2}{\text{fan\_in} + \text{fan\_out}}} \quad \text{(normal)}$$

$$\sigma(W) = \sqrt{\frac{6}{\text{fan\_in} + \text{fan\_out}}} \quad \text{(uniform)}$$

[3] http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

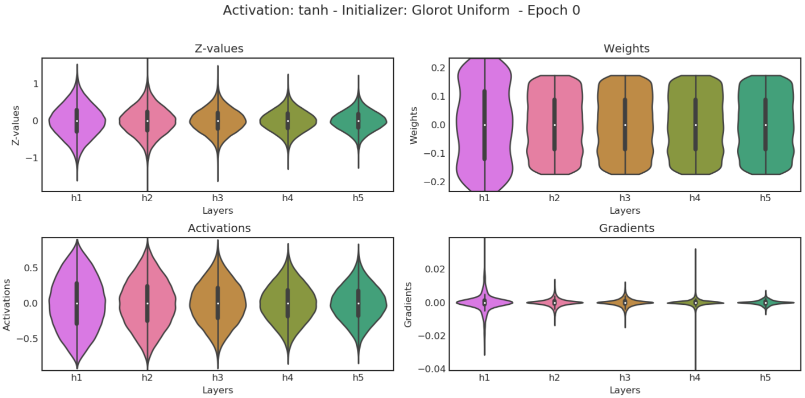
(Xavier) Glorot Initialization



- Normal distribution

Image Credit: Daniel Godoy

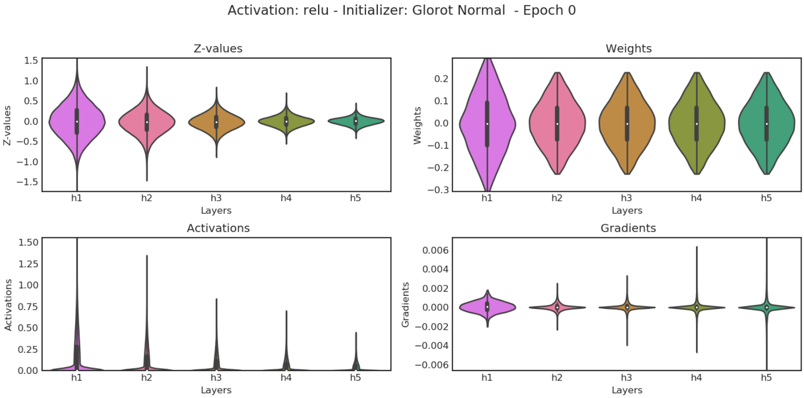
(Xavier) Glorot Initialization



- Uniform distribution

Image Credit: Daniel Godoy

(Xavier) Glorot Initialization



- Does it work for ReLU?

Image Credit: Daniel Godoy

(Kaiming) He Initialization[4]

Carnegie Mellon

- Forward

$$\sigma(W) = \sqrt{\frac{2}{\text{fan\_in}}} \quad \text{(normal)} \qquad \sigma(W) = \sqrt{\frac{6}{\text{fan\_in}}} \quad \text{(uniform)}$$

- Backward

$$\sigma(W) = \sqrt{\frac{2}{\text{fan\_out}}} \quad \text{(normal)} \qquad \sigma(W) = \sqrt{\frac{6}{\text{fan\_out}}} \quad \text{(uniform)}$$

- Combined:

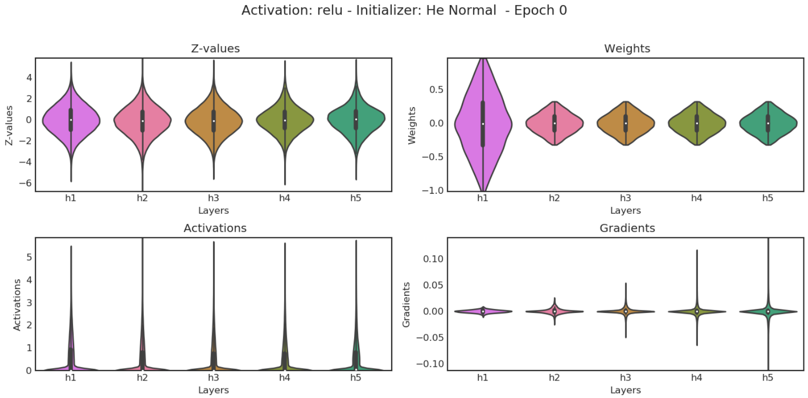
$$\sigma(W) = \sqrt{\frac{4}{\text{fan\_in} + \text{fan\_out}}} \quad \text{(normal)}$$

$$\sigma(W) = \sqrt{\frac{12}{\text{fan\_in} + \text{fan\_out}}} \quad \text{(uniform)}$$

[4] https://arxiv.org/pdf/1502.01852.pdf

(Kaiming) He Initialization

Carnegie Mellon

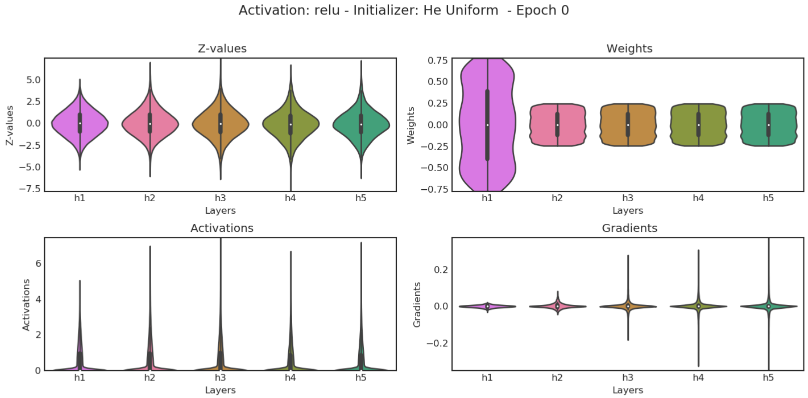


- Normal Distribution

Image Credit: Daniel Godoy

(Kaiming) He Initialization

Carnegie Mellon



- Uniform Distribution

Image Credit: Daniel Godoy

(Kaiming) He Initialization

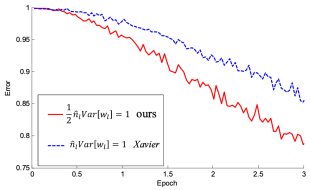


Figure 2. The convergence of a **22-layer** large model (B in Table 3). The x-axis is the number of training epochs. The y-axis is the top-1 error of 3,000 random val samples, evaluated on the center crop. We use ReLU as the activation for both cases. Both our initialization (red) and "*Xavier*" (blue) [7] lead to convergence, but ours starts reducing error earlier.

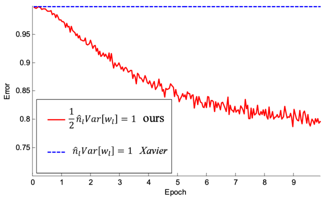


Figure 3. The convergence of a **30-layer** small model (see the main text). We use ReLU as the activation for both cases. Our initialization (red) is able to make it converge. But "*Xavier*" (blue) [7] completely stalls - we also verify that its gradients are all diminishing. It does not converge even given more epochs.

- Better than Glorot inits in their settings

Image Credit: Kaiming He *et al.*

(Kaiming) He Initialization

Carnegie Mellon

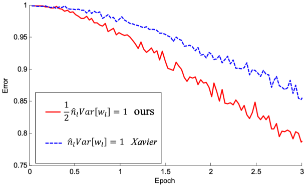


Figure 2. The convergence of a **22-layer** large model (B in Table 3). The x-axis is the number of training epochs. The y-axis is the top-1 error of 3,000 random val samples, evaluated on the center crop. We use ReLU as the activation for both cases. Both our initialization (red) and "*Xavier*" (blue) [7] lead to convergence, but ours starts reducing error earlier.

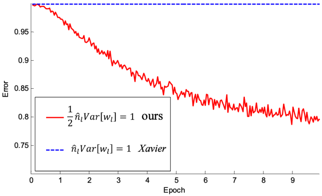
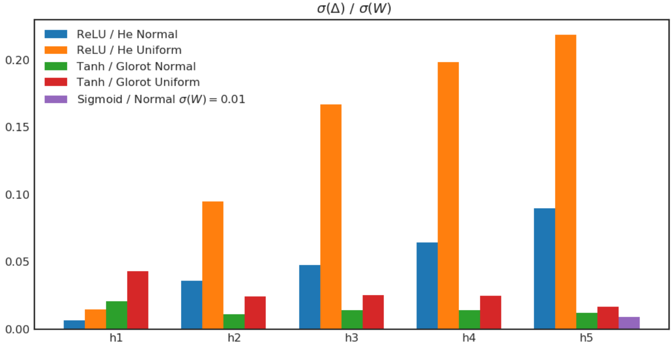


Figure 3. The convergence of a **30-layer** small model (see the main text). We use ReLU as the activation for both cases. Our initialization (red) is able to make it converge. But "*Xavier*" (blue) [7] completely stalls - we also verify that its gradients are all diminishing. It does not converge even given more epochs.

- Better than Glorot inits in their settings
- Many in the community agree, too!

Image Credit: Kaiming He *et al.*

What is the best?



- It varies
- Glorot works well with sigmoid or tanh
- He works well with ReLU and is usually used in vision

Image Credit: Daniel Godoy

More about initialization

Carnegie Mellon

tf.keras.layers.Conv2D example

```
__init__(
    filters,
    kernel_size,
    strides=(1, 1),
    padding='valid',
    data_format=None,
    dilation_rate=(1, 1),
    activation=None,
    use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros',
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

- API: tf.keras.initializers

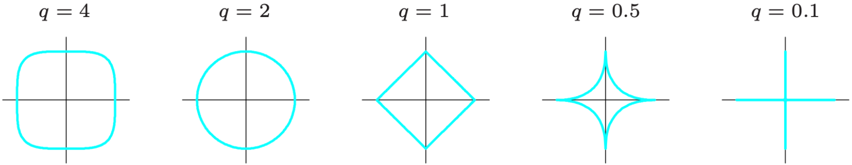
Outline

Carnegie Mellon

1. Initialization
2. Weights regularization
3. Dropout

L-norm Regularization

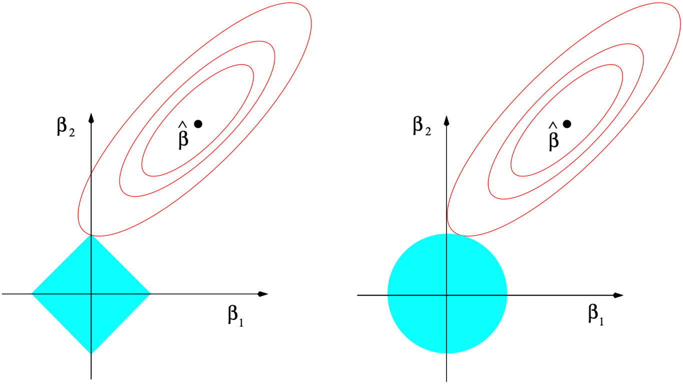
CarnegieMellon



- Is common in practice which helps generalization
- Inject weights constraint and optimize the loss within that constraint
- Incorporate domain prior knowledge (recall: MAP)
- Can penalize based on Lq-norm of the weights: $\|\theta\|_q$

Image Credit: Trevor Hastie *et al.*

L1 & L2 Regularization

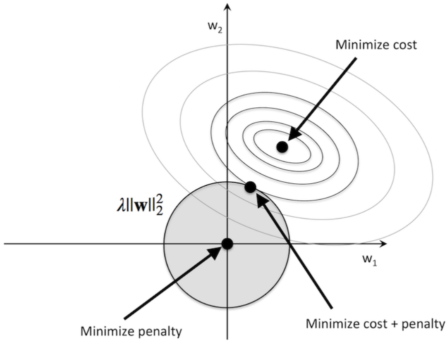


- L1 and L2 are simple, classical but usually works well in practice

Image Credit: Trevor Hastie *et al.*

L2 Regularization

Carnegie Mellon

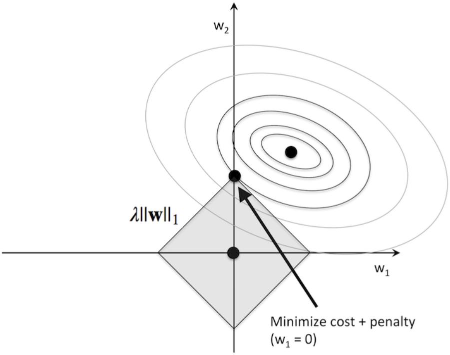


- L2: Loss change: $\frac{\lambda}{2}\|\theta\|_2^2$, Gradients change: $\eta\lambda\theta$
- Analytically sound, and so very popular in practice

Image Credit: MLxtend (Sebastian Rashka)

L1 Regularization

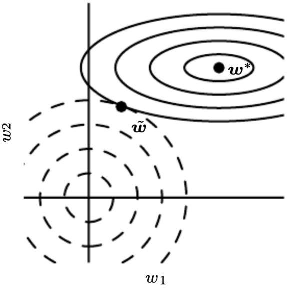
Carnegie Mellon



- L1: Loss change: $\frac{\lambda}{2}\|\theta\|_1$, (sub)Gradients change: $\eta\lambda\,\mathrm{sign}(\theta)$
- Makes weights sparse, *unlike* L2

Image Credit: MLxtend (Sebastian Rashka)

Weight Decaying



- Weight decaying: $\theta^{(t+1)} \leftarrow \alpha\theta^{(t)} - \eta\nabla_\theta l(\theta^{(t)})$ with small α
- L2 regularization: $\tilde{l}(\theta^{(t)}) = l(\theta^{(t)}) + \frac{\lambda}{2}\|\theta^{(t)}\|_2^2$
- They are the same for SGD but not for other adaptive methods
- AdamW[5] implements weight decaying properly for Adam

[5] https://arxiv.org/pdf/1711.05101.pdf

Image Credit: Ian Goodfellow *et al.*

Regularizer in tf.keras

tf.keras.regularizers example

```
# init object
my_l1 = tf.keras.regularizers.l1(l=0.01)
my_l2 = tf.keras.regularizers.l2(l=0.02)
my_l1l2 = tf.keras.regularizers.l1_l2(l1=0.01, l2=0.01)  # L1 + L2 penalties

dense = tf.keras.layers.Dense(kernel_initializer='ones',
                              kernel_regularizer=my_l1,
                              bias_regularizer=my_l2, ... )

conv = tf.keras.layers.Conv2D(kernel_regularizer=my_l2,
                              activity_regularizer=my_l1l2, ...)
```

- Normally, pass a `tf.keras.regularizers` object into a NN layer

Custom Regularizer in tf.keras[6]

Carnegie Mellon

custom_regularizer

```
@tf.keras.utils.register_keras_serializable(package='Custom', name='l2')
class L2Regularizer(tf.keras.regularizers.Regularizer):
  def __init__(self, l2=0.):  # pylint: disable=redefined-outer-name
    self.l2 = l2

  def __call__(self, x):
    return self.l2 * tf.math.reduce_sum(tf.math.square(x))

  def get_config(self):
    return {'l2': float(self.l2)}

dense = tf.keras.layers.Dense(kernel_initializer='ones',
                              kernel_regularizer=L2Regularizer(l2=0.5),
                              ... )
```

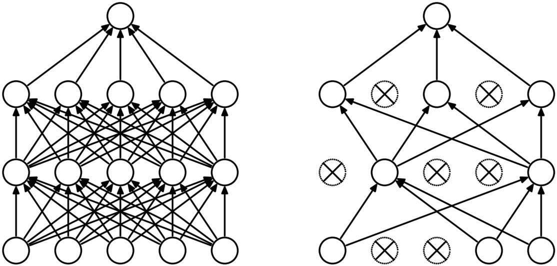
- Need to overwrite `__call__` function properly

[6] https://www.tensorflow.org/api_docs/python/tf/keras/regularizers/Regularizer

Outline

1. Initialization
2. Weights regularization
3. Dropout

Dropout (Hinton)[7]



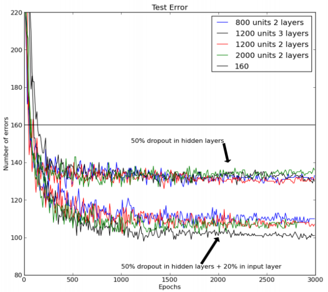
(a) Standard Neural Net (b) After applying dropout.

- One of the most efficient regularization techniques
- Idea: Randomly drop neurons during training to reduce overfitting

[7] http://www.cs.toronto.edu/~hinton/absps/dropout.pdf

Image Credit: Nitish Srivastava *et al.*

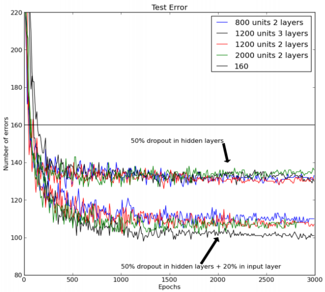
Dropout



- Dropping probability: $\Lambda \sim Ber(p)$ (note: some use keeping prob)
- Training: each activation is changed as follow (both directions):
  - Origin: $h_i = \phi(W_i^T h_{i-1} + b_i)$
  - With dropout: $\Lambda_i \sim Ber(p), \quad h_i = \phi\left(W_i^T \Lambda_i h_{i-1} + b_i\right)$

Image Credit: Geoffrey Hinton *et al.*

Dropout

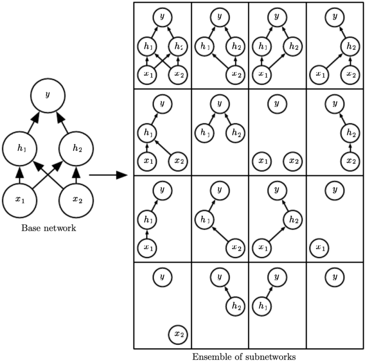


- Dropping probability: $\Lambda \sim Ber(p)$ (note: some use keeping prob)
- Training: each activation is changed as follow (both directions):
  - Origin: $h_i = \phi(W_i^T h_{i-1} + b_i)$
  - With dropout: $\Lambda_i \sim Ber(p), \quad h_i = \phi\left(W_i^T \Lambda_i h_{i-1} + b_i\right)$
- Test: dropout is turned off, and *scaled* (reduced) by a factor of p

Image Credit: Geoffrey Hinton *et al.*

Interpretation: Ensemble

Carnegie Mellon



- Has ensembling effect
- Difference:
  - Exponentially subnetworks, only a small fraction gets trained
  - All subnetworks share different subsets of params

Image Credit: Ian Goodfellow *et al.*

Interpretation: Bayesian Prior[9]

- Noise distribution (*e.g.* Bernoulli[8] as above): $\lambda_i \sim p(\lambda)$
- Noise diagonal matrix: $\Lambda_i = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_{|h_{i-1}|})$
- With dropout: $h_i = \phi\left(W_i^T \Lambda_i h_{i-1} + b_i\right)$
- Loglikelihood is changed to: Expectation of Loglikelihood:

$$\mathcal{L}(\mathbf{X}, \mathbf{y}, \{W_i\}_{i=1}^n) = \mathbb{E}_{p(\lambda)}\left[\log p\left(\mathbf{y} \mid \mathbf{X}, \{W_i\}_{i=1}^n, \{\Lambda_i\}_{i=1}^{n-1}\right)\right]$$

- The *only* assumption: Gaussian prior for all weights

$$W_i \sim \mathbf{N}(\mathbf{0}, \sigma^2)$$

[8] Others such as Gaussian, Half-Cauchy, ... also work well: `https://arxiv.org/pdf/1806.05975.pdf`

[9] `https://arxiv.org/pdf/1810.04045.pdf`

Gaussian Scale Mixtures (GSM)[10]

- Independent scalar random variable: $\lambda \sim p(\lambda)$
- A Gaussian variable with zero mean: $W \sim \mathbf{N}(\mathbf{0}, \sigma^2)$
- What is the distribution of λW?

[10] `https://www.jstor.org/stable/2984774`

Gaussian Scale Mixtures (GSM)[10]

- Independent scalar random variable: $\lambda \sim p(\lambda)$
- A Gaussian variable with zero mean: $W \sim \mathbf{N}(\mathbf{0}, \sigma^2)$
- What is the distribution of λW?

$$\theta \stackrel{d}{=} \lambda W \sim \mathbf{N}(\mathbf{0}, \lambda^2 \sigma^2)$$

[10] https://www.jstor.org/stable/2984774

Gaussian Scale Mixtures (GSM)[10]

Carnegie Mellon

- Independent scalar random variable: $\lambda \sim p(\lambda)$
- A Gaussian variable with zero mean: $W \sim \mathbf{N}(\mathbf{0}, \sigma^2)$
- What is the distribution of λW?

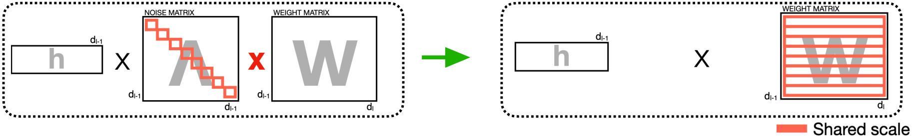
$$\theta \overset{d}{=} \lambda W \sim \mathbf{N}(\mathbf{0}, \lambda^2 \sigma^2)$$

- Super-Gaussian distributions can be represented as GSMs such as horsehoe, student-t, Laplace.

[10] https://www.jstor.org/stable/2984774

Prior Interpretation & Scale Sharing

Carnegie Mellon



- Review the output with dropout:

$$h_i = \phi\left(\underbrace{W_i^T \Lambda_i}_{\text{scale mixtures}} \quad h_{i-1} + b_i\right)$$

$$= \phi\left(\widehat{W_i}^T \quad h_{i-1} + b_i\right) \qquad \text{(original type)}$$

with $\widehat{W}_{i,j} \sim \mathbf{N}(\mathbf{0}, \lambda_i^2 \sigma^2)$, and $\lambda_i \sim p(\lambda_i)$

- This interpretation concurs with Automatic Relevance Determination (ARD) by Mackay[11] and Neal[12]

[11] https://bayes.wustl.edu/MacKay/pred.pdf

[12] http://www.cs.toronto.edu/pub/radford/thesis.pdf

Image Credit: Eric Nalisnick