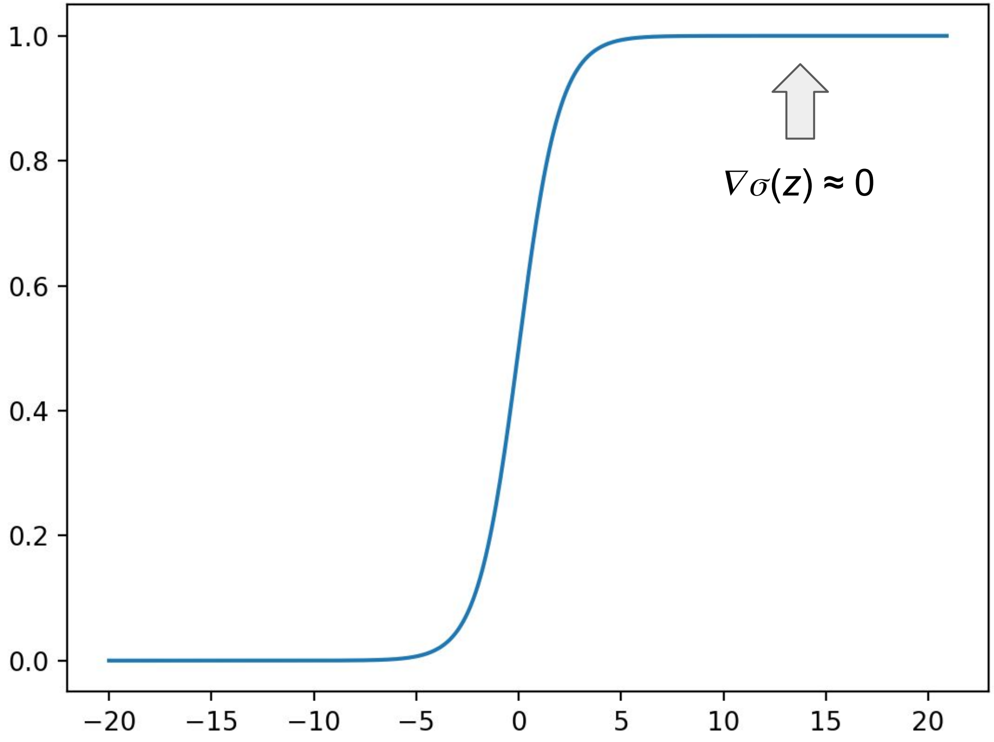
Vanishing Gradients

Chris Barrick

Vanishing Gradients



$\nabla \sigma(z) \approx 0$

Problems with logistic activation

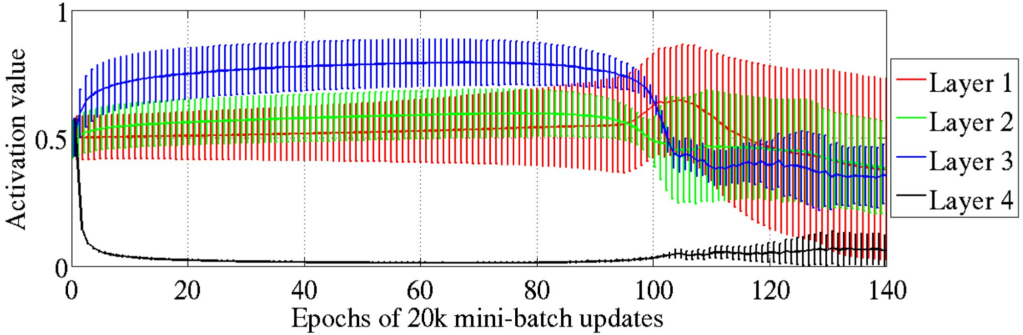


Figure 2: *Mean and standard deviation (vertical bars) of the activation values (output of the sigmoid) during supervised learning, for the different hidden layers of a deep architecture. The top hidden layer quickly saturates at 0 (slowing down all learning), but then slowly desaturates around epoch 100.*

Understanding the difficulty of training deep feedforward neural networks (2010) — Xavier Glorot and Yoshua Bengio

Problems with tanh activation

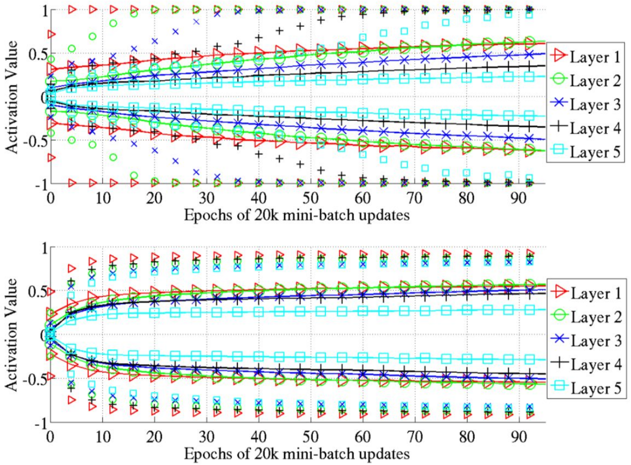


Figure 3: *Top:98 percentiles (markers alone) and standard deviation (solid lines with markers) of the distribution of the activation values for the hyperbolic tangent networks in the course of learning. We see the first hidden layer saturating first, then the second, etc. Bottom: 98 percentiles (markers alone) and standard deviation (solid lines with markers) of the distribution of activation values for the softsign during learning. Here the different layers saturate less and do so together.*

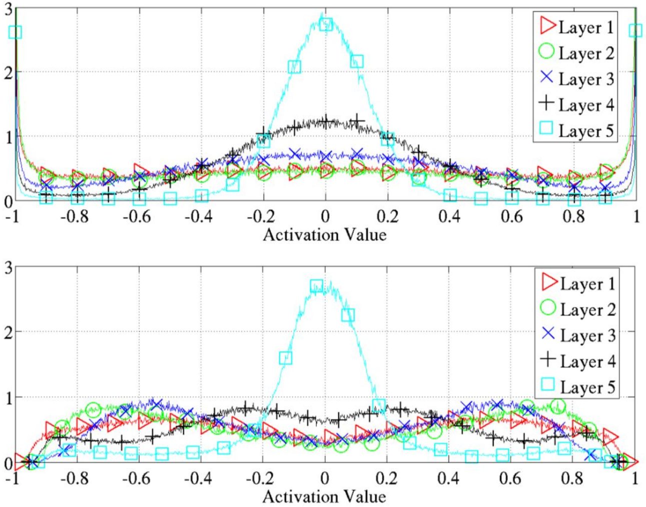


Figure 4: *Activation values normalized histogram at the end of learning, averaged across units of the same layer and across 300 test examples. Top: activation function is hyperbolic tangent, we see important saturation of the lower layers. Bottom: activation function is softsign, we see many activation values around (-0.6,-0.8) and (0.6,0.8) where the units do not saturate but are non-linear.*

Understanding the difficulty of training deep feedforward neural networks (2010) — Xavier Glorot and Yoshua Bengio

Problems with squared error

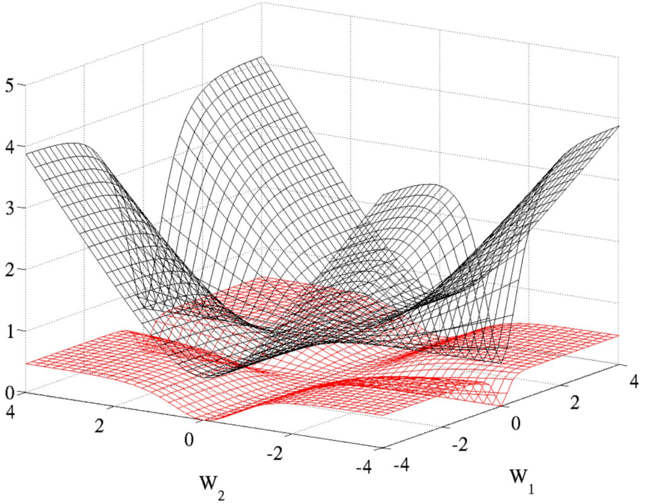


Figure 5: *Cross entropy (black, surface on top) and quadratic (red, bottom surface) cost as a function of two weights (one at each layer) of a network with two layers, W_1 respectively on the first layer and W_2 on the second, output layer.*

Understanding the difficulty of training deep feedforward neural networks (2010) — Xavier Glorot and Yoshua Bengio

Xavier Initialization

$$\forall(i, i'), Var[z^i] = Var[z^{i'}].$$

$$\forall(i, i'), Var\left[\frac{\partial Cost}{\partial s^i}\right] = Var\left[\frac{\partial Cost}{\partial s^{i'}}\right].$$

$$\forall i, \quad n_i Var[W^i] = 1$$

$$\forall i, \quad n_{i+1} Var[W^i] = 1$$

$$\forall i, \quad Var[W^i] = \frac{2}{n_i + n_{i+1}}$$

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

Understanding the difficulty of training deep feedforward neural networks (2010) — Xavier Glorot and Yoshua Bengio

Xavier Initialization

$$\forall(i, i'), Var[z^i] = Var[z^{i'}].$$

$$\forall(i, i'), Var\left[\frac{\partial Cost}{\partial s^i}\right] = Var\left[\frac{\partial Cost}{\partial s^{i'}}\right].$$

$$\forall i, \quad n_i Var[W^i] = 1$$

$$\forall i, \quad n_{i+1} Var[W^i] = 1$$

$$\forall i, \quad Var[W^i] = \frac{2}{n_i + n_{i+1}}$$

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

Understanding the difficulty of training deep feedforward neural networks (2010) — Xavier Glorot and Yoshua Bengio

Xavier Initialization

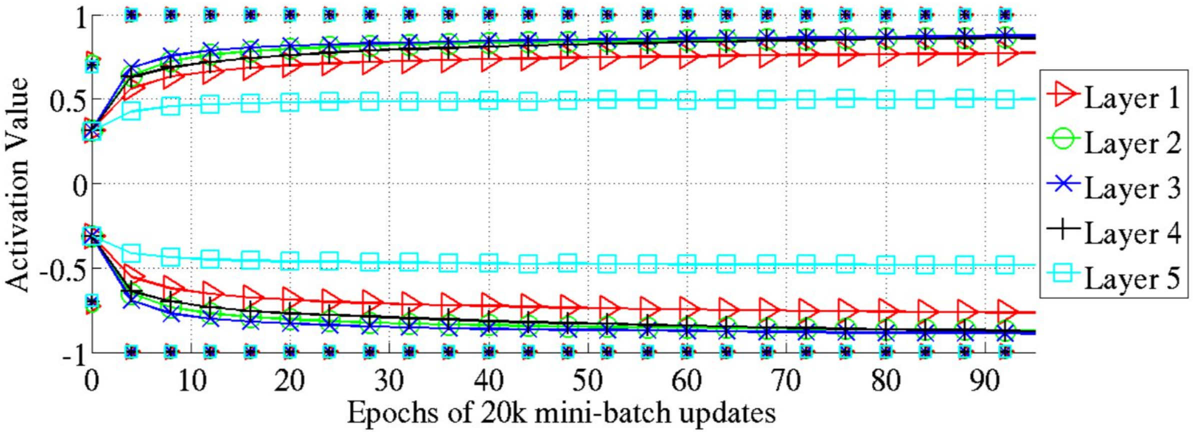


Figure 10: *98 percentile (markers alone) and standard deviation (solid lines with markers) of the distribution of activation values for hyperbolic tangent with normalized initialization during learning.*

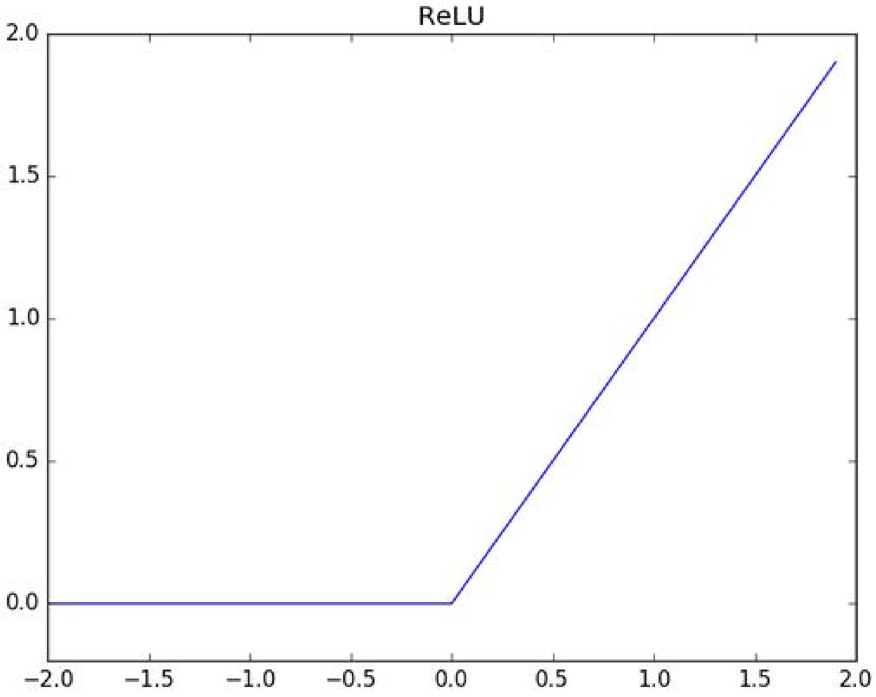
Understanding the difficulty of training deep feedforward neural networks (2010) — Xavier Glorot and Yoshua Bengio

Takeaways from *Understanding the Difficulty*

- The vanishing gradient problem is really about maintaining variance.
- Pay attention to your activations during training.
- Don’t use logistic activation for deep learning.
- Use cross-entropy loss when possible.
- Maintain variance (flow of information) between layers.
- Xavier initialization.

Understanding the difficulty of training deep feedforward neural networks (2010) — Xavier Glorot and Yoshua Bengio

ReLU



Kaiming Initialization

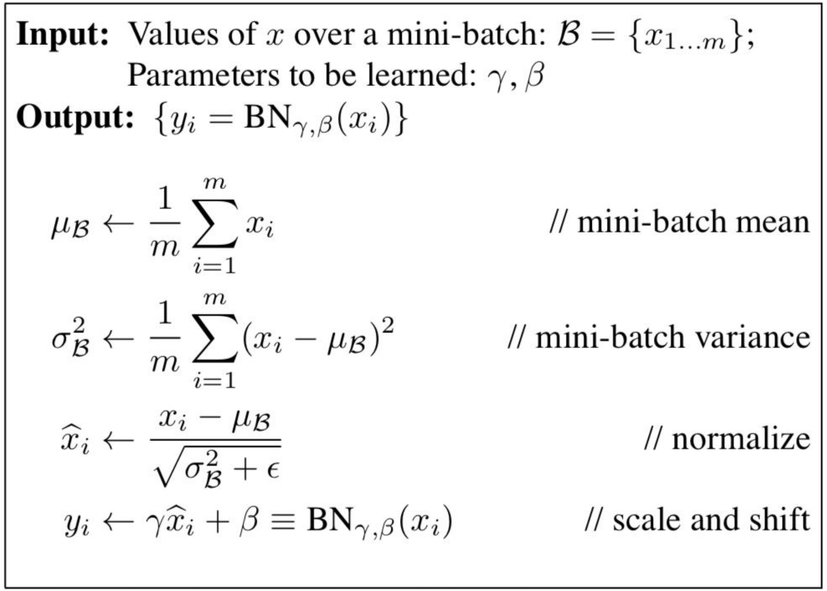
- Reproduced the analysis of Xavier initialization for ReLU and PReLU.
- Only differs by a factor that depends on the activation function.
- Allowed them to train a 30-layer network.

$$\frac{1}{2} n_l Var[w_l] = 1, \quad \forall l.$$

Delving Deep into Rectifiers (2015) — Kaiming He et al.

Batch Normalization

- How do we maintain variance *after* initialization?
- Re-normalize after every activation!
- Reduces the dependence of the gradient on the scale of the parameters.
- “Internal covariate shift.”
- Difficult:
    - Normalize each input independently.
    - Use mini-batch statistics.
    - Apply a learned linear transform afterwards.



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

    $\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$                          // mini-batch mean
    $\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2$     // mini-batch variance
    $\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}$                // normalize
    $y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$         // scale and shift
```

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

Batch Normalization (2015) — Sergey Ioffe and Christian Szegedy

Exponential Linear Unit (ELU)

- “Bias shift” is caused when layers have non-zero mean activation.
- Reducing bias shift brings the gradient closer to the “unit natural gradient”.
- Either re-center activations per layer (e.g. batch norm) or use an activation with negative values.
- ELU outperforms ReLU + batch norm.

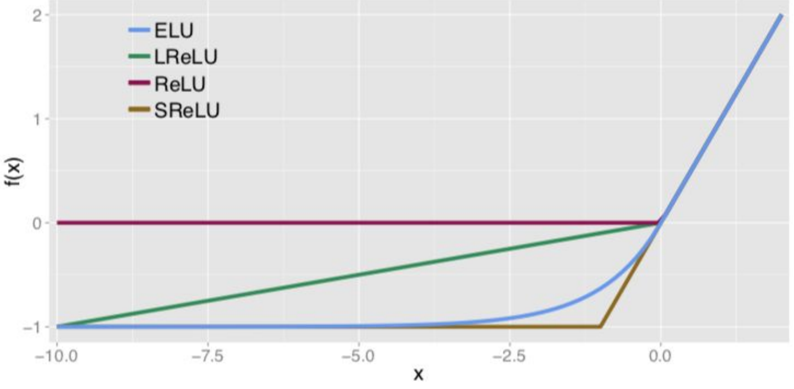
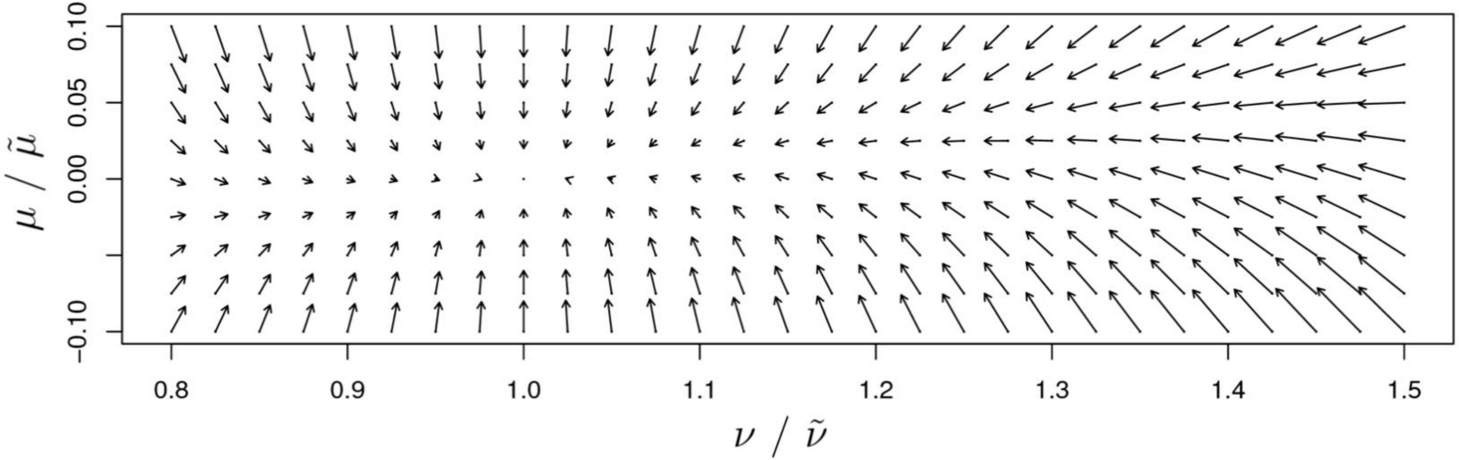


Figure 1: The rectified linear unit (ReLU), the leaky ReLU (LReLU, $\alpha = 0.1$), the shifted ReLUs (SReLUs), and the exponential linear unit (ELU, $\alpha = 1.0$).

Fast and Accurate Deep Network Learning by Exponential Linear Units (2016) — Djork-Arné Clevert et al.

Self-Normalizing Neural Networks

- Parameterized ELU to “Scaled ELU”.
- Defined formal criteria for a network to maintain zero mean, unit variance.
- Solved for the parameters of SELU to meet this criteria.
- Introduced alpha-dropout which maintains mean and variance.



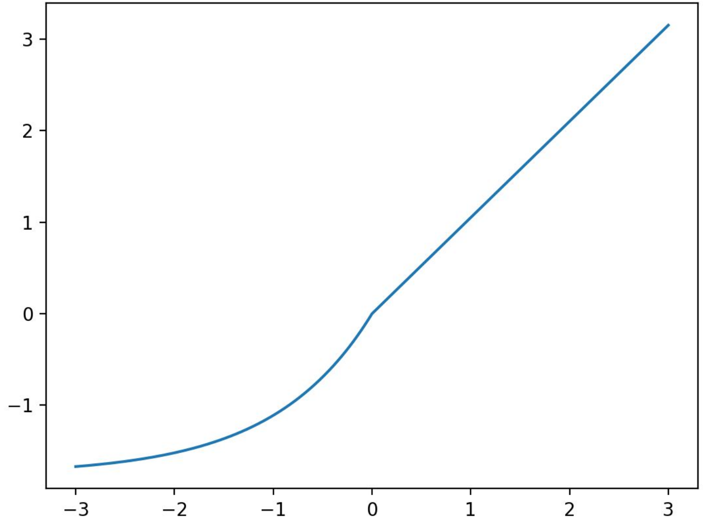
Self-Normalizing Neural Networks (2017) — Günter Klambauer et al.

Self-Normalizing Neural Networks

$$\mathrm{selu}(x) \;=\; \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leqslant 0 \end{cases} .$$

$$\alpha_{01} = -\frac{\sqrt{\frac{2}{\pi}}}{\mathrm{erfc}\left(\frac{1}{\sqrt{2}}\right)\exp\left(\frac{1}{2}\right) - 1} \approx 1.67326$$

$$\lambda_{01} = \left(1 - \mathrm{erfc}\left(\frac{1}{\sqrt{2}}\right)\sqrt{e}\right)\sqrt{2\pi}$$



Self-Normalizing Neural Networks (2017) — Günter Klambauer et al.