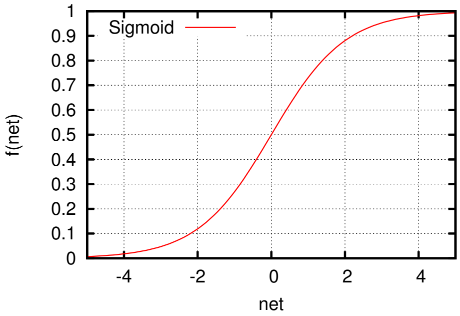
# Supervised Learning Performance Issues

## [Part II]

21 August 2019

# Activation functions
## Are some better than others?



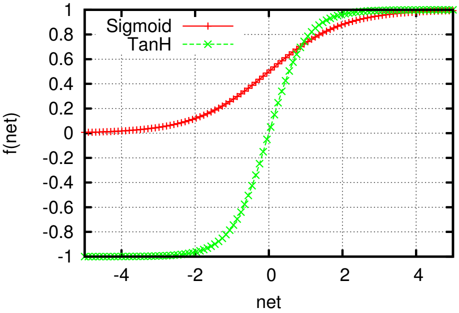

### Sigmoid

$$f(net) = \frac{1}{1 + e^{-net}}$$

- Output can be interpreted as "binary"
- Closest to the original step function
- Range $(0, 1)$: the output will always be positive
- Mean output will not be zero
- What happens to the next layer?..

# Activation functions

## Are some better than others?



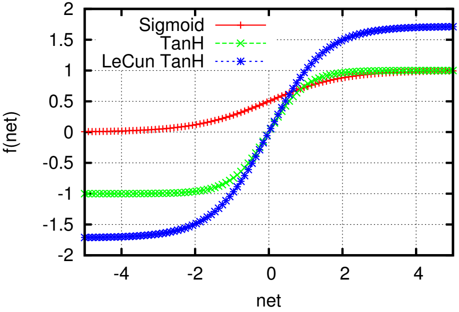

### Hyperbolic tangent (tanh)

$$f(net) = \frac{e^{net} - e^{-net}}{e^{net} + e^{-net}}$$

- Range $(-1, 1)$
- More likely to have the mean output of zero
- Now the outputs are “centred”
- Less chance of subsequent saturation
- Empirically shown to converge faster
- What about the variance?

# Activation functions

"Efficient Backprop", Y. LeCun et al., 1998



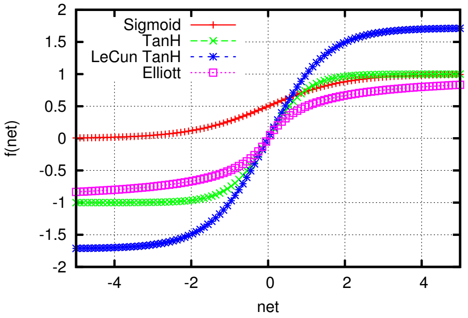

## Yann LeCun tanh

$$f(net) = 1.7159 \tanh\left(\frac{2}{3} net\right)$$

- Range $(-1.7159, 1.7159)$
- ???
- The constants were derived by LeCun to ensure that the variance of outputs is 1
- Now the outputs are centred around zero with a variance of one
- I.e., they are standardized!

# Activation functions

"A Better Activation Function for Artificial Neural Networks", D.L. Elliott, 1993



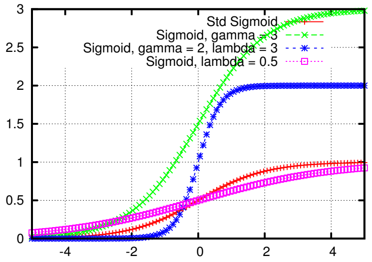

## Elliott

$$f(net) = \frac{net}{1+|net|}$$

- Range $(-1, 1)$
- Easier to compute than Sigmoid
- Can be scaled to achieve variance of 1
- Softer slope than TanH
  - Slower learning
  - Less saturation

# Engelbrecht's adaptive activation function

## Learn the function slope together with the weights

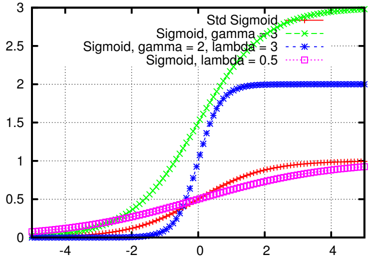



### Lambda-Gamma Learning

- Learn the steepness and range of $f(net)$
- $f(net, \lambda, \gamma) = \frac{\gamma}{1+e^{-\lambda net}}$
- $\lambda$ determines the slope steepness
- $\gamma$ determines the range
- Slope/range is learned => no need for input data scaling
- Is there a catch?

# Engelbrecht's adaptive activation function

## Learn the function slope together with the weights



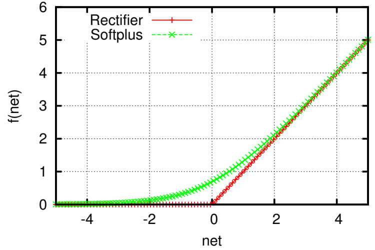

### Lambda-Gamma Learning

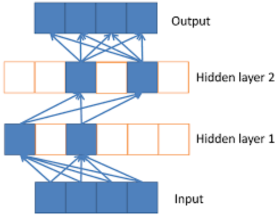
- Update the training algorithm:
- $o_k = f(net, \lambda_{o_k}, \gamma_{o_k})$
- $\delta_{o_k} = -\frac{\lambda_{o_k}}{\gamma_{o_k}}(t_k - o_k)o_k(\gamma_{o_k} - o_k)$
- $\lambda_{o_k} = \lambda_{o_k} + \eta_2 \delta_{o_k} \frac{net_{o_k}}{\lambda_{o_k}}$
- $\gamma_{o_k} = \gamma_{o_k} + \eta_3 (t_k - o_k) \frac{1}{\gamma_{o_k}} o_k$
- Have to choose values for $\eta_2$ and $\eta_3$ in addition to $\eta_1$

# Deep Learning Activation: Rectified Linear Unit

## ReLU: Biologically plausible?

"Deep Sparse Rectifier Neural Networks", Glorot et al, 2011




**Rectifier**

$$f(net) = \max(0, net)$$

**Softplus**

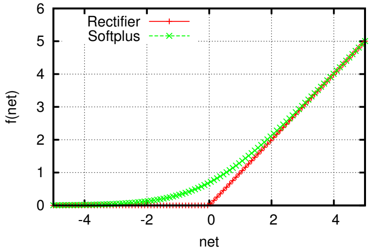
$$f(net) = \log(1 + e^{net})$$

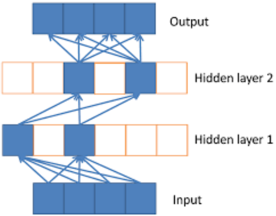
- No gradient when $f(net) = 0$
- Gradient is 1 when $f(net) > 0$
- No vanishing gradient!
- Sparse activations ($\approx 50\%$)
- Non-linearity is achieved via different "paths" being activated

# Deep Learning Activation: Rectified Linear

## Biologically plausible?

“Deep Sparse Rectifier Neural Networks”, Glorot et al, 2011




**Rectifier**

$$f(net) = \max(0, net)$$

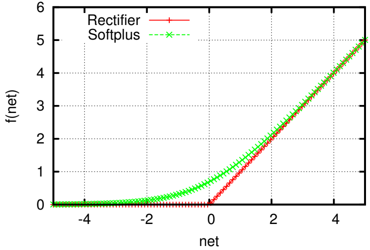
**Softplus**

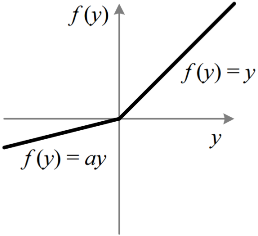
$$f(net) = \log(1 + e^{net})$$

- Problems?
- Non-zero centered
- Unbounded
- Saturated ReLUs “die”

# Rectifier

"Delving Deep into Rectifiers: Surpassing Human-Level Performance on Image Net Classification", He et al, 2015




## Rectifier

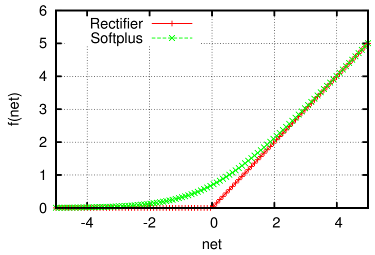
$$f(net) = \max(0, net)$$

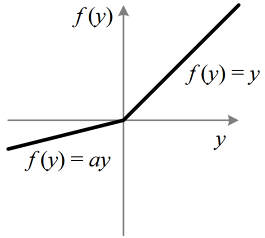
## Leaky/Parametrised Rectifier

$$f(net) = \begin{cases} net & \text{if } net > 0 \\ a * net & \text{otherwise} \end{cases}$$

- Original paper: $a = 0.01$
- Parametrised: learn the value of $a$
- $\Delta a_{i+1} = \alpha \Delta a_i + \eta \frac{\partial E}{\partial a_i}$

# Batch Normalisation

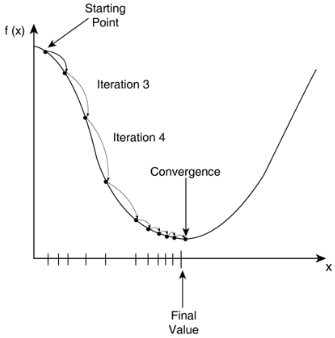



- How do we remedy skew activations?
- "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", Ioffe & Szegedy, 2015
- Cheat the system: standardise activations
- $\hat{f}(net_i) = \frac{f(net_i) - \bar{f}(net)}{\sigma f(net)}$
- $y_i = \gamma \hat{f}(net_i) + \beta$
- Values of $\gamma$ and $\beta$ are learned per layer

# Backpropagation Parameters

Tuning the most popular NN training algorithm




## Learning Rate and Momentum

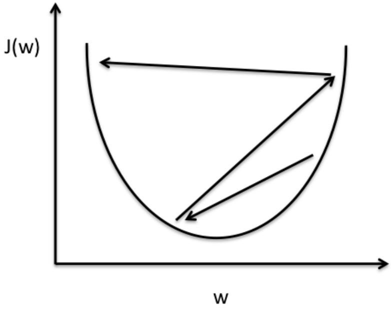
- Stochastic Backprop:
- $w_t := w_t + \Delta w_t + \alpha \Delta w_{t-1}$
- $\Delta w_t = \eta(-\frac{\partial E}{\partial w_t})$
- $\alpha$ - momentum; controls the influence of past weight changes on the current weight change
- $\eta$ - learning rate; controls the magnitude of the step size
- How do we choose values for $\eta$ and $\alpha$?

# Effect of Learning Rate on Training

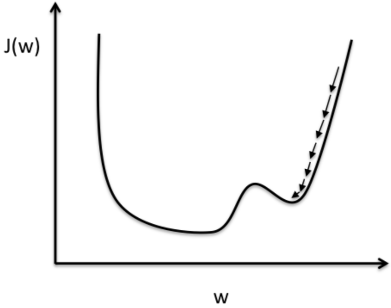
## Learning Rate

- Stochastic Backpropagation algorithm:
- $w_t = w_t + \Delta w_t + \alpha \Delta w_{t-1}$
- $\Delta w_t = \eta(-\frac{\partial E}{\partial w_t})$
- If $\eta$ is small, step size will be small
  - Search path will closely resemble the gradient path
  - Learning will be slow
- If $\eta$ is large, step size will be large
  - Might skip over good regions
  - Learning will be fast

# Effect of Learning Rate on Training
## Learning Rate




**Large learning rate: Overshooting.**




**Small learning rate: Many iterations until convergence and trapping in local minima.**

# Choosing the Learning Rate

## Choosing $\eta$

- Cross-validation: try a selection of values, choose the best-performing one
- Start with a small value (0.1), increase if convergence is slow, decrease if oscillation/stagnation is observed
- Plaut et al: $\eta \sim \frac{1}{fanin}$ (more weights => smaller steps)
- Every $w_i$ can have its own $\eta_i$
  - If direction of change (i.e. sign of $\Delta w_i$) has not changed since previous weight change, increase $\eta_i$ (go faster)
  - Else, decrease $\eta_i$ (go slower)
- Decaying $\eta$: First explore, then exploit
  - Half-life: divide by 2 every 5 epochs
  - ...Or shrink in some other way every $n$ iterations
  - Larger -> smaller

# Effect of Momentum on Training

## Momentum term

- Stochastic Backpropagation algorithm:
- $w_t = w_t + \Delta w_t + \alpha \Delta w_{t-1}$
- $\Delta w_t = \eta(-\frac{\partial E}{\partial w_t})$
- Stochastic learning: adjust weights after each pattern
- Result: sign of the error derivative fluctuates, making the NN “unlearn” what it has learned in the previous steps
- Solution: Batch learning
- Alternatively: add momentum to the equation - average the weight changes as you go, maintain direction
- Larger $\alpha$ => direction of $\Delta w_t$ must be preserved for longer to affect the direction of weight changes
- Would this be necessary with batch, mini-batch learning?

# Choosing the Momentum

## Choosing $\alpha$

- Use a static value of 0.9
- Cross-validation: try a selection of values, choose the best-performing one
- Adaptive $\alpha$: start with a smaller value (0.5), increase over time (0.9)
- Every $w_i$ can have its own $\alpha_i$
  - Follow a quadratic approximation of the previous gradient step and the current gradient
  - Quickprop (Fahlman):
  - $\alpha_i(t) = \frac{\frac{\partial E}{\partial w_i(t)}}{\frac{\partial E}{\partial w_i(t-1)} - \frac{\partial E}{\partial w_i(t)}}$
  - Becker & LeCun (scale each weight by curvature):
  - $\alpha = \left(\frac{\partial^2 E}{\partial w_i^2(t)}\right)^{-1}$ (Becker & LeCun)

# Modern Adaptive Methods

http://ruder.io/optimizing-gradient-descent/index.html

## Momentum and learning rate are not independent

- Larger momentum allows larger step sizes
- Good strategy:
  - Set momentum to as high a value as possible (0.999?)
  - Choose the largest convergent learning rate

## AdaGrad: adaptive learning rate per weight

- Idea: make rare (sparse) events count more
  - $\nabla w_t = \frac{\partial E}{\partial w_t}$
  - $s_{w_t} = s_{w_{t-1}} + (\nabla w_t)^2$
  - $\Delta w_t = -\frac{\eta}{\sqrt{s_{w_t} + \epsilon}} \nabla w_t$
- High gradients -> smaller update
- Smaller/infrequent gradients -> larger update
- Over the course of training, $\Delta w_t \to 0$

# Modern Adaptive Methods

http://ruder.io/optimizing-gradient-descent/index.html

## AdaDelta / RMSProp (Resilient mean squared propagation)

- AdaGrad variation: prevent $\Delta w_t$ from decaying to zero:
  - $\nabla w_t = \frac{\partial E}{\partial w_t}$
  - Exponentially decaying avg: $s_{w_t} = d * s_{w_{t-1}} + (1-d)(\nabla w_t)^2$
  - $d = 0.9$, can be optimised
  - $\Delta w_t = -\frac{\eta}{\sqrt{s_{w_t} + \epsilon}} \nabla w_t$

## Adam

- Combine RMSProp with momentum:
  - $\nabla w_t = \frac{\partial E}{\partial w_t}$
  - $m_{w_t} = d_1 * m_{w_{t-1}} + (1-d_2)\nabla w_t$
  - $s_{w_t} = d_2 * s_{w_{t-1}} + (1-d_2)(\nabla w_t)^2$
  - $d_1 = 0.9, d_2 = 0.999$
  - $\Delta w_t = -\frac{\eta}{\sqrt{s_{w_t} + \epsilon}} m_{w_t}$

# The End

- Questions?
- No lecture next week!
- Assignment 1 is available. Due date: 4 September
- Lecture on 4 September: Architecture selection