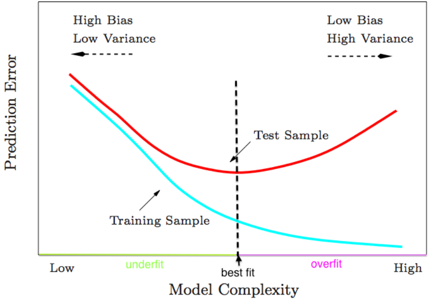
Activation Functions

Nan Ye

School of Mathematics and Physics
The University of Queensland

What is Needed for Generalization?

- Recall

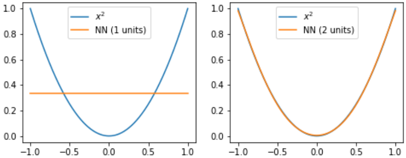


- To attain good generalization performance
  - the model class need to be able to approximate the true model well.
  - in addition, it should be possible to *learn* a (near-) optimal approximation within the class.
- We look at how activation functions affect the generalization performance of neural nets in this lecture.

Approximation Power

- True input-output relationship: $y = f(x) = x^2$, $x \in [-1, 1]$.
- Assumed model: single-hidden layer MLP, sigmoid activation for hidden units, and identity activation for the output unit.
- Is there a model that approximates f well?
  - *i.e. can we find a neural net* $g(x)$ *of the form* $\sum_{i=1}^{m} \alpha_i \sigma(w_i x + b_i) + \beta$ *such that* $|g(x) - f(x)|$ *is small for all* $x \in [-1, 1]$*?*

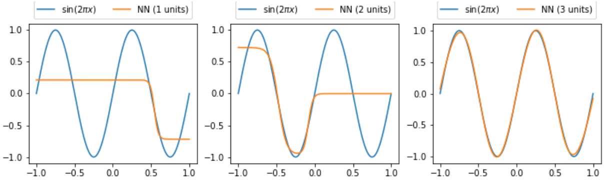
- Yes, we just need 2 sigmoid units for a very good approximation!



$$x^2 \approx 2.2\sigma(-3.15x - 3) + 2.2\sigma(3.15x - 3) - 0.205,\ x \in [-1, 1].$$

- What happens if $f(x) = \sin(2\pi x)$?

- We just need 3 sigmoid units for a very good approximation!



$$\sin(x) \approx 10.9\sigma(-6.35x - 3.05) - 10.9\sigma(6.35x - 3.05) - 36.6\sigma(-1.3x) + 18.23,$$
$$x \in [-1, 1].$$

- Why can neural nets approximate these functions well?

- A key factor is the activation function
  - Why? MLPs with identity function are just linear functions $\Rightarrow$ they can't approximate x^2 or $\sin(2\pi x)$ well.
  - So the sigmoid activation plays an important role in our examples.
- Can we approximate functions other than x^2 and $\sin(2\pi x)$? Does other activation functions work?
- Universal approximation theorems give affirmative answers to these questions.

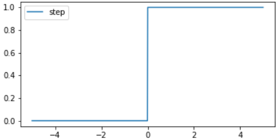
Universal Approximation

- Under mild conditions, a single-hidden layer MLP using a bounded, continuous and monotonically increasing activation function have the universal approximation property
  - universal approximation = with sufficiently many neurons, we can approximate any continuous function arbitrarily well (typically on a compact domain)
  - Example activations: sigmoid, tanh
- This can be extended to certain unbounded activation functions
  - Example activations: ReLU, truncated power
- Most activations are thus "equal" in the sense that they have the universal approximation property.

- However, they are not really equal considering
  - computational efficiency
  - difficulty to optimize
  - generalization performance
- This motivates much research on designing good activation functions.

Binary Step

- $f(u) = \begin{cases} 1, & u \geq 0. \\ 0, & u < 0. \end{cases}$



- Used in the perceptron (using -1 instead of 0 to denote inactive state), but provably hard to train in general.

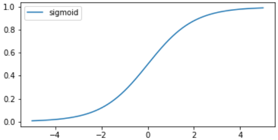
+ biologically appealing, as biological neurons generate all-or-none electrochemical pulses.

– it makes the neural net discontinuous, thus not ideal for approximating continuous continuous functions.

– saturates too easily ⇒ hard to learn
  - saturation = little/no change if input increases further
  - saturation ⇒ vanishing gradients ⇒ gradient-based learning is hard
  - Gradient, if exists, is always 0 for binary step activation!

Sigmoid (aka logistic)

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

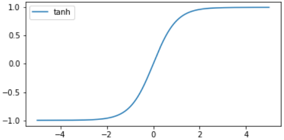


sigmoid squashes input to the range (0,1)

- + it is continuous and differentiable with smoothly changing gradients $\Rightarrow$ gradient-based learning is possible
- – it still saturates for large inputs, and this kill the gradients.
- – exponentiation is a bit expensive
- – sigmoid outputs are not zero-centered
  - *recall: we use various tricks to keep input zero-centered*

tanh (hyperbolic tangent)

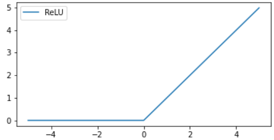
- $f(u) = \tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$



  tanh squashes numbers to the range (-1,1).
- Essentially the same pros and cons as sigmoid, but tanh is zero-centered.
- Note: both sigmoid and tanh are used in LSTM (why? we need to take the desirable range of the output into account).

ReLU

- $f(u) = \max(0, u) = (u)_+$



The ReLU unit does not saturate in +region.

- It approximates the sum of infinitely many shifted copies of the logistic unit.

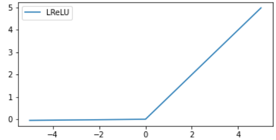
$$\sum_{i=1}^{\infty} \sigma(x - i + 0.5) \approx \ln(1 + e^x) \approx \max(0, x).$$

Nair and Hinton, Rectified linear units improve restricted boltzmann machines, 2010

+ ReLU is efficiently computable (no exponentiation), and converges much faster than sigmoid/tanh in practice.
+ ReLU often has better generalization than sigmoid/tanh in practice.
− output not zero-centered.
− a dead ReLU never activates again.

Leaky ReLU (LReLU)

- $f(u) = \max(0.01u, u)$.



  Leaky ReLU does not saturate.
- Leaky ReLU has essentially the same pros and cons as ReLU, but it never dies.

Maas, Hannun, and Ng, Rectifier nonlinearities improve neural network acoustic models, 2013

Parametric ReLU (PReLU)

- Slopes other than 0.01 may be used for the negative part in leaky ReLU.
- Tuning the slope using cross-validation can lead to better results, but too tedious and slow.
- A better idea: learn a domain-specific slope together with other parameters.
- Parametric ReLU

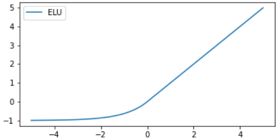
$$f(u) = I(u < 0)\alpha u + I(u \geq 0)u,$$

where $\alpha > 0$ is learnable parameter.

He, Zhang, Ren, and Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015

ELU (Exponential Linear Unit)

- $f(u) = (u)_+ + \min(0, \alpha(e^u - 1))$ ($\alpha > 0$ is user-specified).

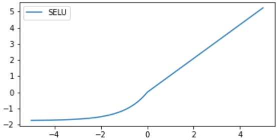


+ ELU alleviates vanishing gradients in the positive part as in ReLU/LReLU/PReLU.
+ It has closer to zero mean outputs than ReLU.
+ Its negative saturation regime adds some robustness to noise as compared to LReLU/PReLU.
− It requires exponentiation.

Clevert, Unterthiner, and Hochreiter, Fast and accurate deep network learning by exponential linear units (elus), 2015

SELU (Scaled ELU)

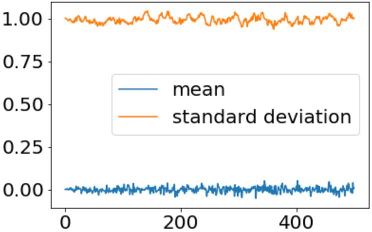
- $f(x) = \lambda[(u)_+ + \min(0, \alpha(e^u - 1))]$ with $\alpha \approx 1.6733$ and $\lambda \approx 1.0507$.



- ELU uses user-defined α and $\lambda = 1$.
- In MLP using SELU, activations converge to zero mean and unit variance given normalized input and weights $\sim N(0, \frac{1}{n_{\text{in}}})$.
  - Normalization tricks are not needed.

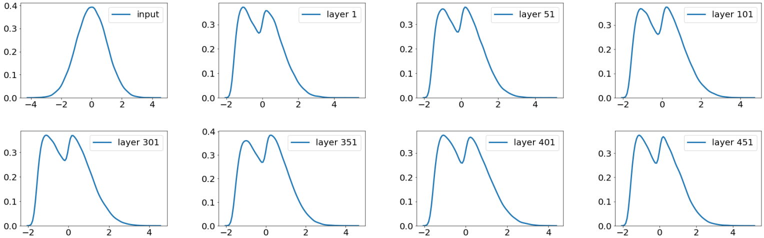
Klambauer, Unterthiner, Mayr, and Hochreiter, Self-normalizing neural networks, 2017

- Plot of mean and variance of activations against number of layers



Note that the mean and variance remain close to 0 and 1 respectively.

- Distribution of the activations are bimodal for normalized input



The distributions remain quite stable.

- For activations not close to unit variance, there is an upper and lower bound on the variance, thus, vanishing and exploding gradients are impossible.

Derivatives

name	$f(x)$	$f'(x)$
sigmoid/logistic	$\frac{1}{1+e^{-u}}$	$f(u)(1-f(u))$
hyperbolic tangent	$\frac{e^u-1}{e^u+1}$	$\frac{1}{2}(1-f(u)^2)$
ReLU	$\max(0,u)$	$I(u>0)$
leaky ReLU	$\max(0.01u,u)$	$0.01I(u\leq 0)+I(u>0)$
parametric ReLU	$\max(\alpha u,u)$	$(\alpha I(u\leq 0)+I(u>0), uI(u\leq 0))$
ELU	$\begin{cases} u, & \text{if } u>0, \\ \alpha(e^u-1), & \text{if } u\leq 0. \end{cases}$	$\begin{cases} 1, & \text{if } u>0, \\ f(u)+\alpha, & \text{if } u\leq 0. \end{cases}$

- Derivative computation is cheap after function evaluation
  - even for activations requiring exponentiation

Choosing an Activation Function

- Avoid exponentiation if time is critical
- Avoid saturating units if training often get stuck
- ReLU is often a good starting point, but fancier versions may give you some performance improvement.
- Zero-mean and unit variance is desirable.

What You Need to Know

- Activation functions often differ wrt computational efficiency, difficulty to optimize, generalization performance.
- Some commonly used activation functions
  - Step, sigmoid, tanh, ReLU, LReLU, PReLU, ELU, SELU
- General rules for choosing an activation function