CAI 4104/6108 – Machine Learning Engineering:

Training Neural Networks

Prof. Vincent Bindschaedler

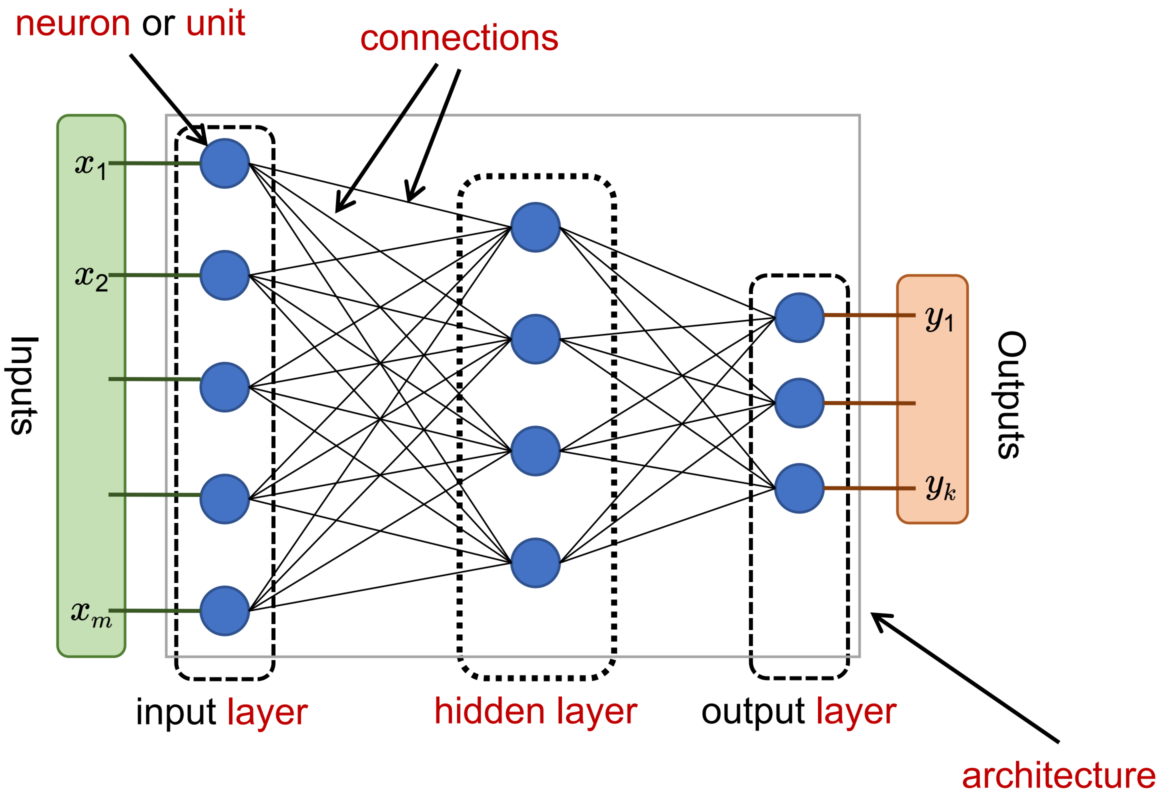
Spring 2024

Administrivia

- Midterm (update on grading)
  - We should be done grading later this week
  - Grades will most likely be out (sometime) next week

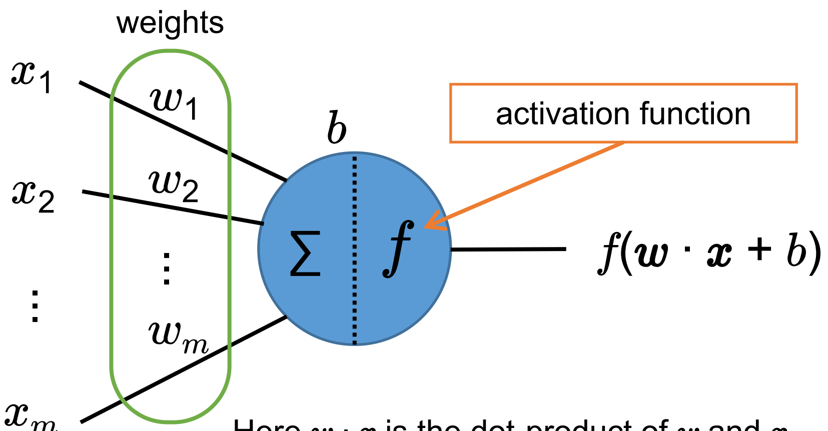
- **No class** on Friday (3/1)
  - Exercise 7 will be **pre-recorded** (available on Canvas)

Reminder: Neural Network Terminology



Reminder: A Simple Neural Network

- Consider a single neuron / unit
  - The model is $h_{\boldsymbol{w},b}(\boldsymbol{x}) = f(\boldsymbol{w} \cdot \boldsymbol{x} + b)$
    - What if we take f to be the identity function?
      - That is: $f(z) = z$
    - What if we take f to be the sigmoid / logistic function?
      - That is: $f(z) = 1/(1+e^{-z})$

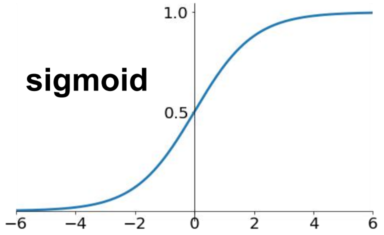


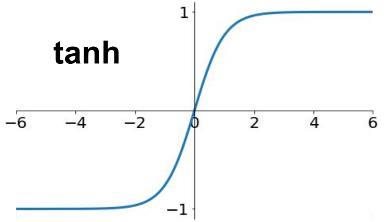
Here $\boldsymbol{w} \cdot \boldsymbol{x}$ is the dot-product of $\boldsymbol{w}$ and $\boldsymbol{x}$.
So $f(\boldsymbol{w} \cdot \boldsymbol{x} + b) = f(\sum w_i x_i + b)$

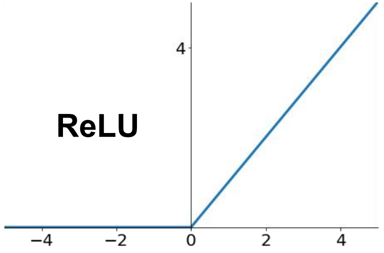
- The Perceptron
  - Invented by Frank Rosenblatt in 1957
    - "The Perceptron—a perceiving and recognizing automaton". Report 85-460-1. Cornell Aeronautical Laboratory
  - A different neuronal architecture called a threshold linear unit (TLU)
    - No bias term
    - With a step activation function. For example:
      - heaviside(z) = 0 if z ≤ 0, 1 otherwise (z ≥ 1) ; or sign(z)

Reminder: Components

- Types of Layers
  - Dense (i.e., fully-connected)
  - Convolutional
  - Recurrent
- Activation Functions
  - Identity / Linear (or none): $f(z) = z$
  - Sigmoid: $f(z) = 1/(1+e^{-z})$
  - TanH: $f(z) = (e^{z} - e^{-z}) / (e^{z} + e^{-z})$
  - ReLU: $f(z) = \max(0, z)$
  - Softmax: $f(z_j) = \exp(z_j / T) / \sum_i \exp(z_i / T)$
    - Note: in that case the activation function is over an entire layer, not a single unit
- Loss
  - Whatever you like (e.g., squared error loss) as long as it's differentiable
    - Note: make sure the loss function and activation function of the output layer are consistent with each other!

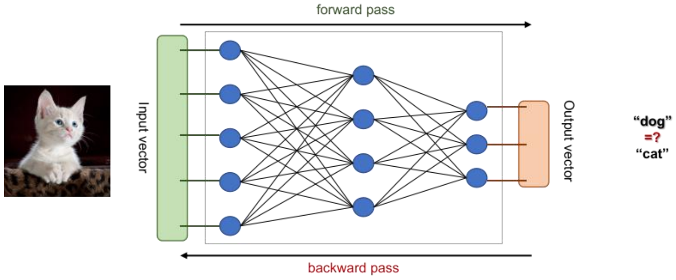






Reminder: Backpropagation

- Seminal Paper:
  - “*Learning representations by back-propagating errors.*” Rumelhart, Hinton, and Williams. Nature 1986.

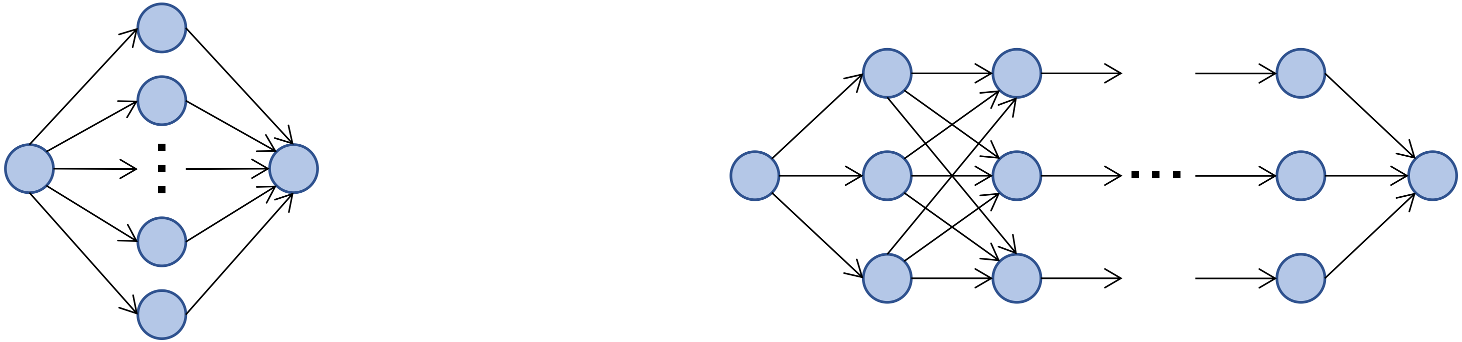


- Terminology
  - Backpropagation: how to compute the gradients efficiently
  - Gradient descent: how to update the parameters to minimize the loss given the gradient

- Algorithm: given a mini-batch B
  - Compute the forward pass for the mini-batch B saving the intermediate results at each layer
  - Compute the loss on the mini-batch B (compares output of network to labels/targets → error)
  - Backwards pass: computes the per-weight gradients (error contribution) layer by layer
    - This is done using the chain rule (if z depends on y and y depends on x: $dz/dx = dz/dy \cdot dy/dx$)
  - (Stochastic) gradient descent: update the weights based on the gradients

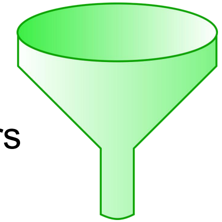
Reminder: Universality of Neural Network

- Universal Approximation Theorems
  - *(Feed-forward) Neural networks can* approximately *represent any function*
  - Arbitrary width; bounded depth:
    - True even if we have a single hidden layer as long as it can have arbitrarily many units
  - Bounded width; arbitrary depth:
    - True even if we have layers of bounded width, as long as the network can have arbitrarily many layer



Neural Network Architecture?

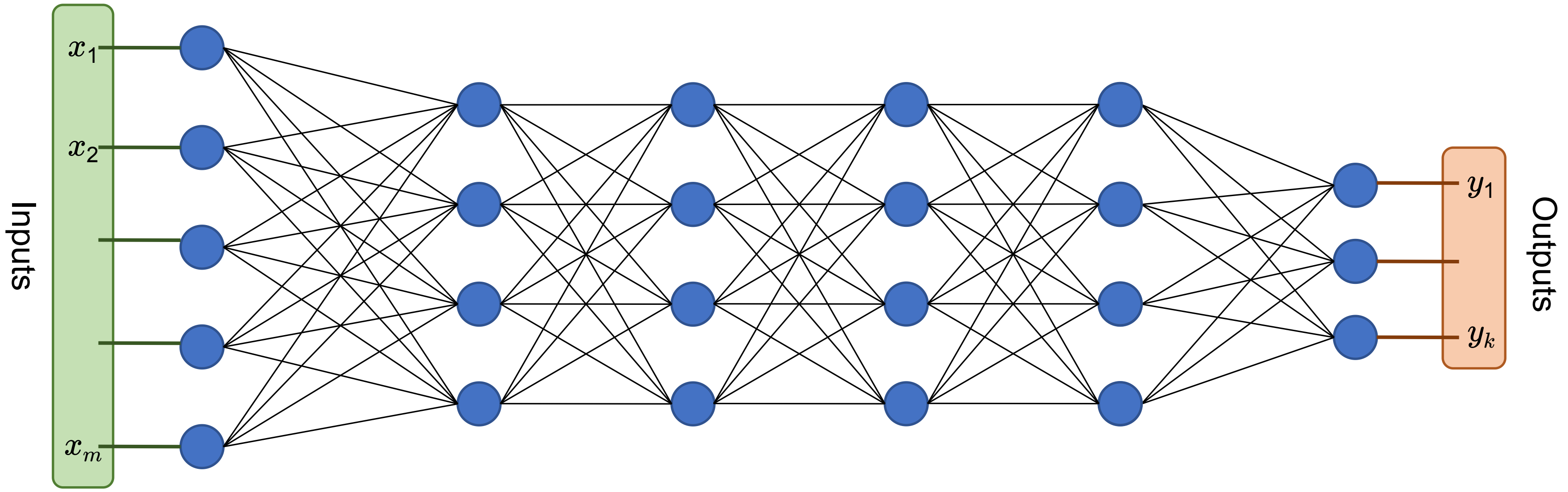
- Pitfall: inconsistent activation function of output layer with the loss function
  - Examples:
    - Multiclass classification with cross-entropy loss, softmax activation for output layer => **Okay**
    - Regression with MSE as loss, tanh as activation for output layer => **Fail**
    - Regression with MSE as loss, linear activation for output layer => **Okay**
- Tip: the “funnel”
  - For supervised learning we typically have large input feature vectors and small output vectors
    - We should make the network look like a funnel



- Example: Multiclass classification with 10 classes and m=100 input features.
  - The network could look like this:
    - (Input, hidden layer 1, hidden layer 2, hidden layer 3, output layer)
      - 100, 64, 32, 16, 10
    - Activations:
      - Output: Softmax
      - Elsewhere: ReLU

Deep Neural Networks

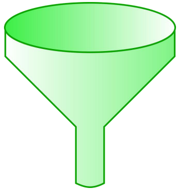
- What is a deep neural network?
  - Any neural network with two or more hidden layers
  - Nowadays, the best neural networks architectures for many applications & problems are deep
    - E.g.: AlexNet (2012) has 8 layers, ResNet18 has 18 layers, GPT-2 has 48 layers



Architecture & Hyperparameters Tuning

- Challenges:
  - Endless options for the network architecture/topology
    - E.g.: number of layers; units per layer; connections between units; activation functions; weight initialization method
  - Hyperparameters related to learning:
    - E.g.: optimizer, learning rate, decay/momentum, (mini)batch size, number of epochs, etc.

- Rules of Thumb:
  - Number of hidden layers
    - ***Deep > shallow***: For the same number of parameters, more hidden layers is better than wider layers
    - Why? Parameter efficiency
  - Number of units in each layer
    - Funnel approach: make the network look like a funnel
    - "Stretch pants" approach: make hidden layers wider than what you need and then regularize (e.g., dropout)
      - Ref: Vanhoucke' Udacity course on Deep Learning

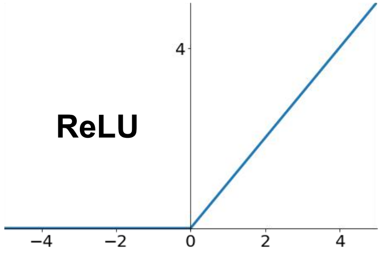


Architecture & Hyperparameters Tuning

- Challenges:
  - Endless options for the network architecture/topology
    - E.g.: number of layers; units per layer; connections between units; activation functions; weight initialization method
  - Hyperparameters related to learning:
    - E.g.: optimizer, learning rate, decay/momentum, (mini)batch size, number of epochs, etc.

- Rules of Thumb:
  - Activation functions
    - Hidden layers => ReLU (or ReLU variants)
      - Faster to compute than alternatives; Gradient descent less likely to get "stuck"
    - Output layer:
      - Multiclass classification: *softmax*
      - Binary classification or multilabel: *sigmoid*
      - Regression: *linear* (no activation function)

ReLU



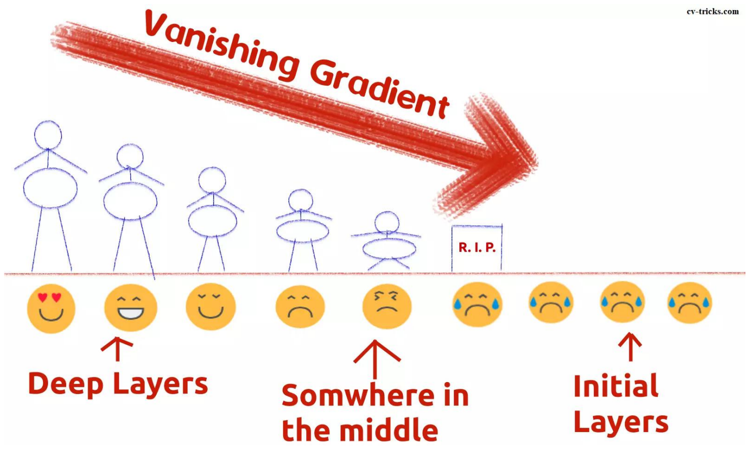
Architecture & Hyperparameters Tuning

- Challenges:
  - Endless options for the network architecture/topology
    - E.g.: number of layers; units per layer; connections between units; activation functions; weight initialization method
  - Hyperparameters related to learning:
    - E.g.: optimizer, learning rate, decay/momentum, (mini)batch size, number of epochs, etc.

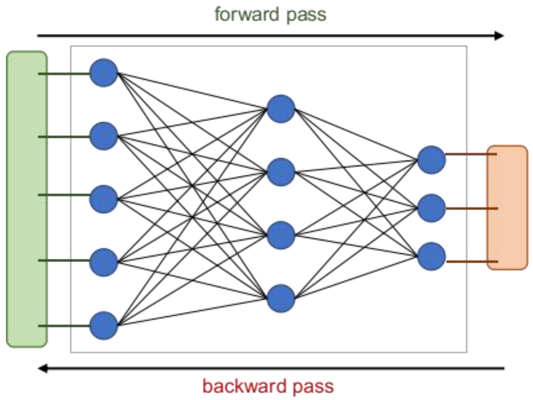
- Rules of Thumb:
  - Learning rate:
    - Start with a low value (e.g., 0.000001) then multiply by 10 each time and train for a few epochs; once training diverges you have gone too far
  - Optimizer: use RMSProp or Adam
  - Batch size:
    - Small batch approach: e.g., 32, 64, 100
    - Large batch approach: the largest size that fits in your GPU's RAM (e.g., 8192) **and** use learning rate warmup
  - Number of epochs/iterations: use early stopping

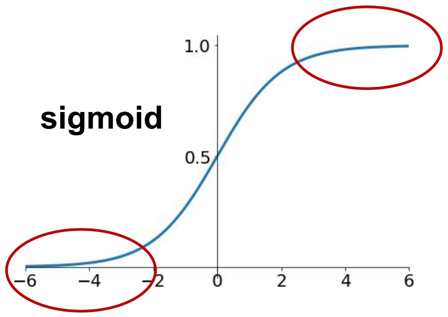
Vanishing Gradient & Exploding Gradient

- Vanishing/Exploding Problems:
  - Gradient vector becomes very small (vanishing gradient) or very large (exploding gradient) during backpropogation
    - Difficult to update weights of lower/earlier layers => Training does not converge
  - Instance of a more general problem: unstable gradients
    - Layers (of a deep neural network) learn at very different rates



Source: https://cv-tricks.com/keras/understand-implement-resnets/

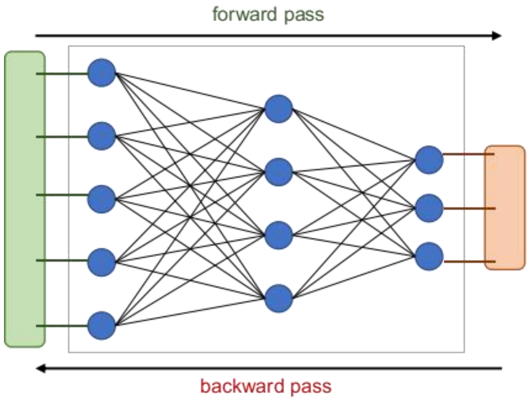




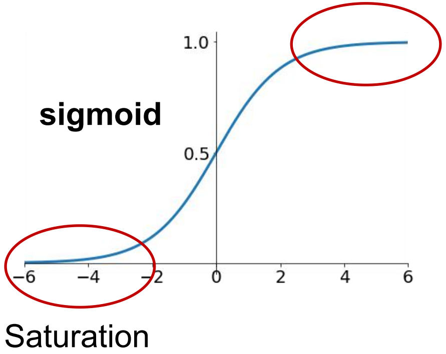
Saturation

Vanishing Gradient & Exploding Gradient

- Vanishing/Exploding Problems:
  - Gradient vector becomes very small (vanishing gradient) or very large (exploding gradient) during backpropogation
    - Difficult to update weights of lower/earlier layers => Training does not converge
  - Instance of a more general problem: unstable gradients
    - Layers (of a deep neural network) learn at very different rates



- Mitigations:
  - Weight initialization method
  - Non-saturating activation functions
  - Batch normalization
  - Gradient clipping (for exploding gradient)
  - Skip-connections (CNNs)
  - ...

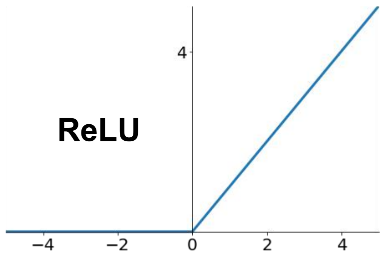


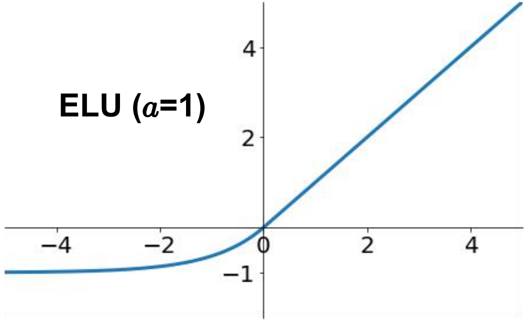
Vanishing Gradient & Exploding Gradient

- Weight Initialization Methods:
  - Some ways to initialize the weights can make gradients unstable
    - This held back efforts to train deep neural networks in the 2000s
  - Glorot initialization (aka Xavier initialization):
    - Seminal paper: Xavier Glorot and Yoshua Bengio. "*Understanding the difficulty of training deep feedforward neural networks*." In AISTATS, 2010.
    - Let n_{in}: number of inputs, n_{out}: number of outputs $n_{\text{avg}} = (n_{\text{in}} + n_{\text{out}})/2$
    - Gaussian (for sigmoid activation): mean 0, variance $\sigma^2 = 1/n_{\text{avg}}$
    - Uniform (for sigmoid activation): in $[-r, r]$ where $r = [3/n_{\text{avg}}]^{1/2}$ [default for Keras]
  - He initialization:
    - Kaiming He et al. "*Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.*" In ICCV, 2015.
    - Gaussian (for ReLU): mean 0, variance $\sigma^2 = 2/n_{\text{avg}}$
  - Note: biases are initialized to 0

Vanishing Gradient & Exploding Gradient

- Non-saturating activation functions
  - Activation functions like ReLU behave better than sigmoid and tanh
  - But ReLU has one problem: dying ReLUs
    - A neuron can "die" when the weighted sums of its input are negative (for all examples in the training data)
    - In this case, ReLU would always output 0
  - ReLU variants:
    - LeakyReLU$_a(z)$ = $\max\{az, z\}$ (e.g., a = 0.01)
    - ELU$_a(z)$ = z if $z \geq 0$ and a (e^z - 1) otherwise ($z < 0$) (e.g., a = 1)
    - There is also SELU (Scaled ELU)
    - These will not let neurons die because they can output negative values
- So what to use in practice?
  - My advice: use ReLU in most cases; if you have extra time use ELU (network will be slower)
  - Although: ELU > leaky ReLU > ReLU > Tanh > Sigmoid





Next Time

- Friday (3/1): **No Class**
  - [*Pre-recorded*] Exercise 7

- Upcoming:
  - Homework 3 will be out soon