Activation function f: the signal is summed and if the final sum is above a certain threshold, the neuron can fire, sending a spike along its axon. In other words, each neuron performs a dot product with the input and its weights, adds the bias and applies the non-linearity (or activation function), in this case the sigmoid $\sigma(x)=1/(1+e^{-x})$.

Single neuron as a linear classifier: a neuron has the capacity to “like” (activation near one) or “dislike” (activation near zero) certain linear regions of its input space. Hence, with an appropriate loss function on the neuron’s output, we can turn a single neuron into a linear classifier.

Binary Softmax classifier: For example, we can interpret $\sigma(\sum_i w_i x_i + b)$ to be the probability of one of the classes $P(y_i=1 \mid x_i;w)$. The probability of the other class would be $P(y_i=0 \mid x_i;w)=1-P(y_i=1 \mid x_i;w)$, since they must sum to one. With this interpretation, we can formulate the cross-entropy loss as we have seen in the Linear Classification section, and optimizing it would lead to a binary Softmax classifier (also known as logistic regression). Since the sigmoid function is restricted to be between 0-1, the predictions of this classifier are based on whether the output of the neuron is greater than 0.5.

Binary SVM classifier. Alternatively, we could attach a max-margin hinge loss to the output of the neuron and train it to become a binary Support Vector Machine.

Regularization interpretation. The regularization loss in both SVM/Softmax cases could in this biological view be interpreted as gradual forgetting, since it would have the effect of driving all synaptic weights w towards zero after every parameter update.

Sigmoid: $\sigma(x)=1/(1+e^{-x})$: it takes a real-valued number and “squashes” it into range between 0 and 1. In particular, large negative numbers become 0 and large positive numbers become 1.

- Drawbacks:
    - when the neuron’s activation saturates at either tail of 0 or 1, the gradient at these regions is almost zero, which can kill the signal flow.
    - sigmoid outputs are not zero centered: This could introduce undesirable zig-zagging dynamics in the gradient updates for the weights. However, notice that once these gradients are added up across a batch of data the final update for the weights can have variable signs, somewhat mitigating this issue.

Tanh. $\tanh(x)=2\sigma(2x)-1$: kike the sigmoid neuron, its activations saturate, but unlike the sigmoid neuron its output is zero-centered. Therefore, in practice the tanh non-linearity is always preferred to the sigmoid nonlinearity. Also note that the tanh neuron is simply a scaled sigmoid neuron.

ReLU: advantages and disadvantages:

- (+) It was found to greatly accelerate (e.g. a factor of 6 in Krizhevsky et al.) the convergence of stochastic gradient descent compared to the sigmoid/tanh functions. It is argued that this is due to its linear, non-saturating form.

- (+) Compared to tanh/sigmoid neurons that involve expensive operations (exponentials, etc.), the ReLU can be implemented by simply thresholding a matrix of activations at zero.
- (-) Unfortunately, ReLU units can be fragile during training and can "die". For example, a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again. If this happens, then the gradient flowing through the unit will forever be zero from that point on. That is, the ReLU units can irreversibly die during training since they can get knocked off the data manifold. For example, you may find that as much as 40% of your network can be "dead" (i.e. neurons that never activate across the entire training dataset) if the learning rate is set too high. With a proper setting of the learning rate this is less frequently an issue.

Leaky ReLU: Leaky ReLUs are one attempt to fix the "dying ReLU" problem. Instead of the function being zero when $x < 0$, a leaky ReLU will instead have a small negative slope (of 0.01, or so). That is, the function computes $f(x)=\mathbb{1}(x<0)(\alpha x)+\mathbb{1}(x>=0)(x)$ where $\alpha$ is a small constant. Some people report success with this form of activation function, but the results are not always consistent.

Maxout: generalizes the ReLU and its leaky version. The Maxout neuron computes the function max(wT1x+b1,wT2x+b2). The Maxout neuron therefore enjoys all the benefits of a ReLU unit (linear regime of operation, no saturation) and does not have its drawbacks (dying ReLU). However, unlike the ReLU neurons it doubles the number of parameters for every single neuron, leading to a high total number of parameters.

Neural Network architectures:

- Layer-wise organization: Neural Networks as neurons in graphs. Neural Networks are modeled as collections of neurons that are connected in an acyclic graph. In other words, the outputs of some neurons can become inputs to other neurons. Cycles are not allowed since that would imply an infinite loop in the forward pass of a network. Instead of an amorphous blobs of connected neurons, Neural Network models are often organized into distinct layers of neurons. For regular neural networks, the most common layer type is the fully-connected layer in which neurons between two adjacent layers are fully pairwise connected, but neurons within a single layer share no connections.
    - Naming conventions. Notice that when we say N-layer neural network, we do not count the input layer. Therefore, a single-layer neural network describes a network with no hidden layers (input directly mapped to output).
    - Output layer. Unlike all layers in a Neural Network, the output layer neurons most commonly do not have an activation function (or you can think of them as having a linear identity activation function). This is because the last output layer is usually taken to represent the class scores (e.g. in classification), which are arbitrary real-valued numbers, or some kind of real-valued target (e.g. in regression).

- Sizing neural networks. The two metrics that people commonly use to measure the size of neural networks are the number of neurons, or more commonly the number of parameters.

Setting number of layers and their sizes: as we increase the size and number of layers in a Neural Network, the capacity of the network increases. That is, the space of representable functions grows since the neurons can collaborate to express many different functions.

Overfitting: occurs when a model with high capacity fits the noise in the data instead of the (assumed) underlying relationship. For example, the model with 20 hidden neurons fits all the training data but at the cost of segmenting the space into many disjoint red and green decision regions.