Data Science

Lecture 5: Deep Learning Overview

University of Amsterdam

Lecturer: Yen-Chia Hsu

Date: Feb 2026

This lecture will give a high-level overview of deep learning and artificial neurons.

Deep learning is the idea of stacking different types of layers (e.g., artificial neurons) to perform very complex tasks. The example below is a deep feedforward neural network.

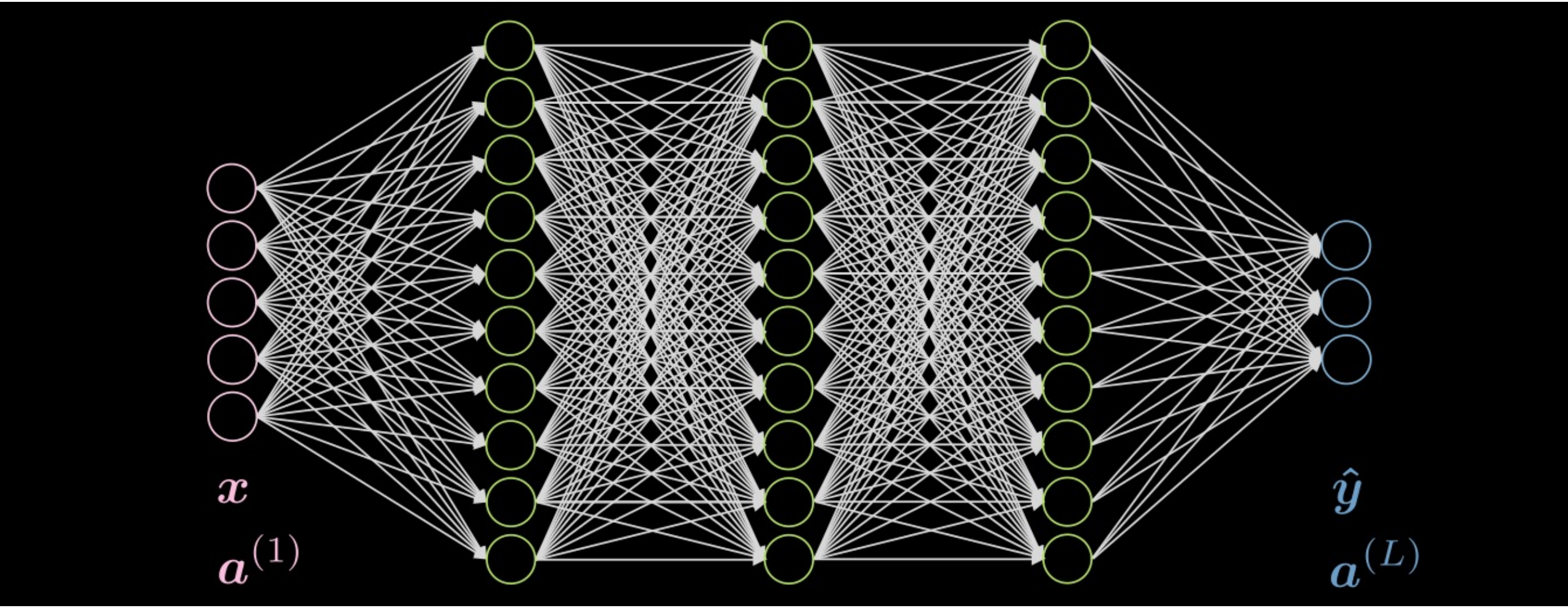
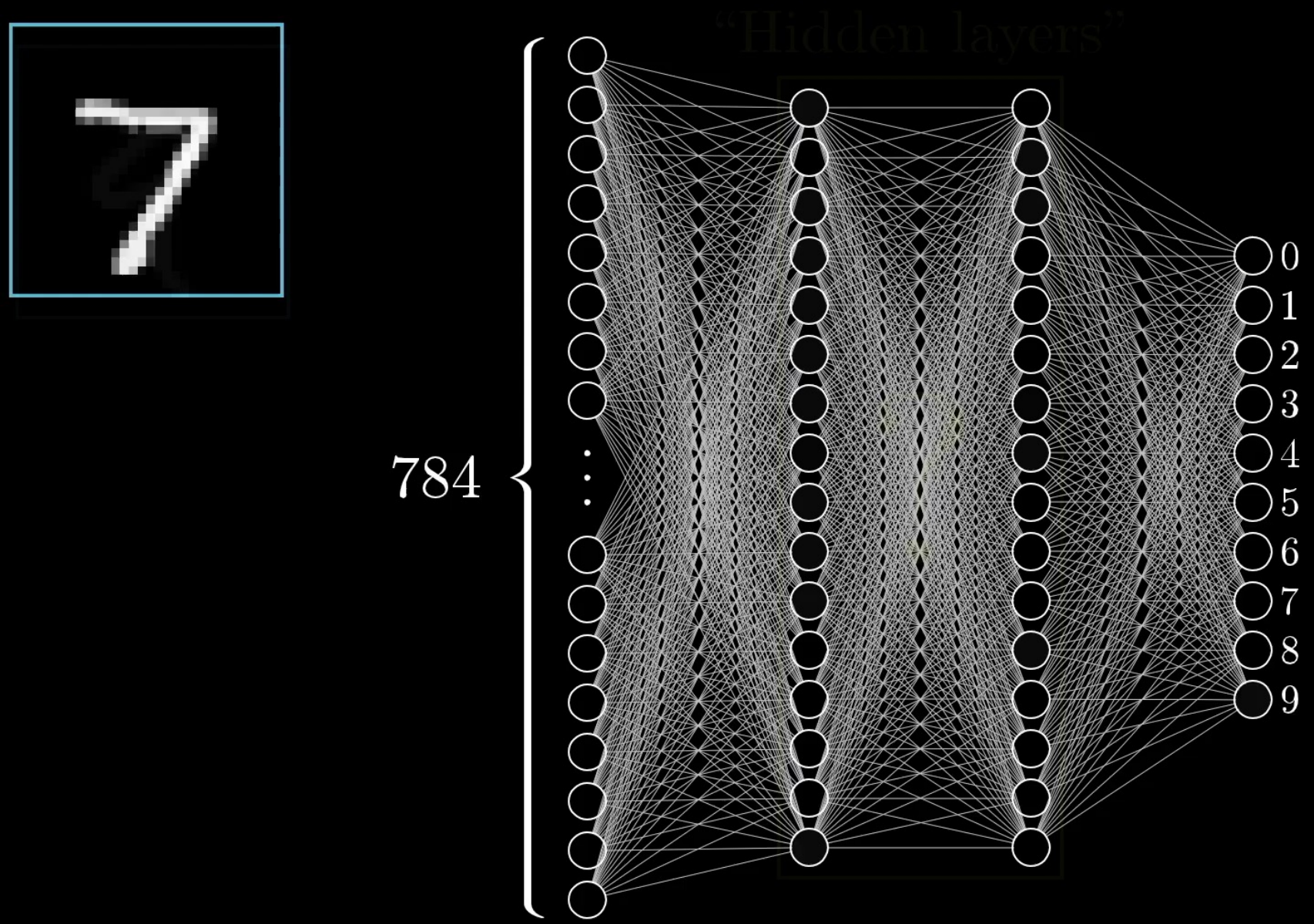


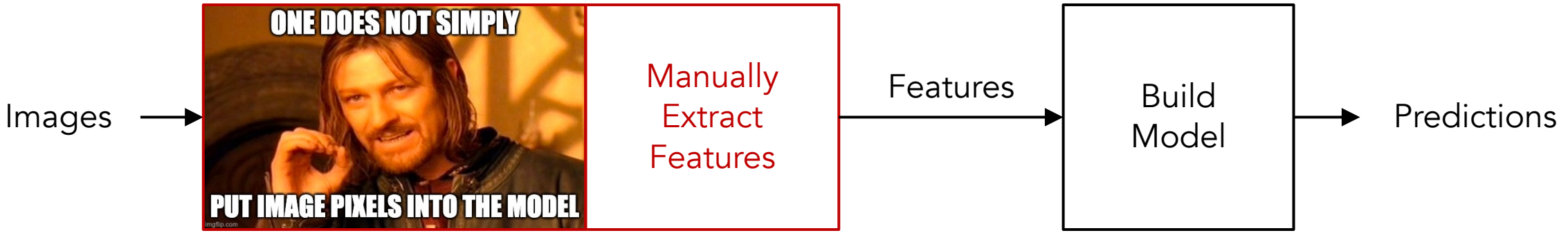
Image source -- https://atcold.github.io/pytorch-Deep-Learning/en/week02/02-3/

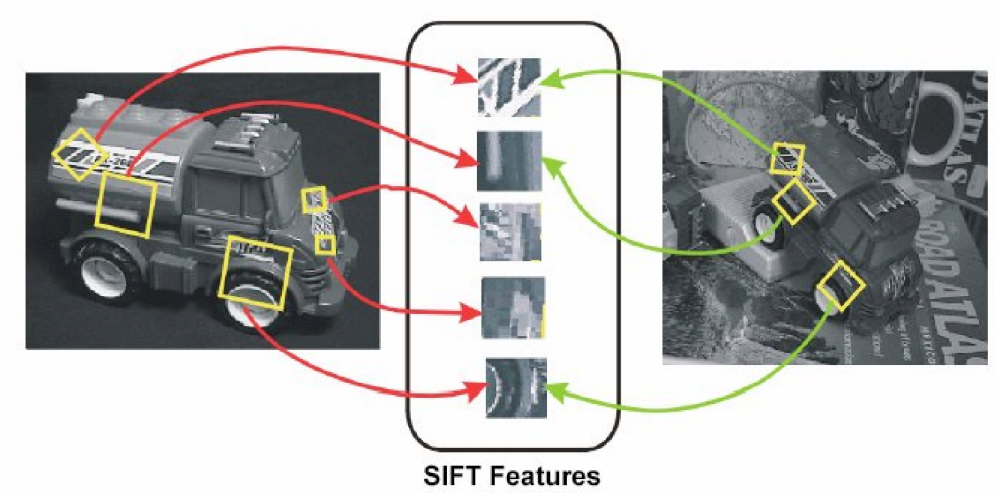
https://www.3blue1brown.com/lessons/neural-networks



Neurons in the first layer are activated based on the input. Then, neurons are activated layer by layer.

Before the deep learning era, machine learning researchers need to extract features from the data manually. But now, we can delegate feature engineering to the neural net.





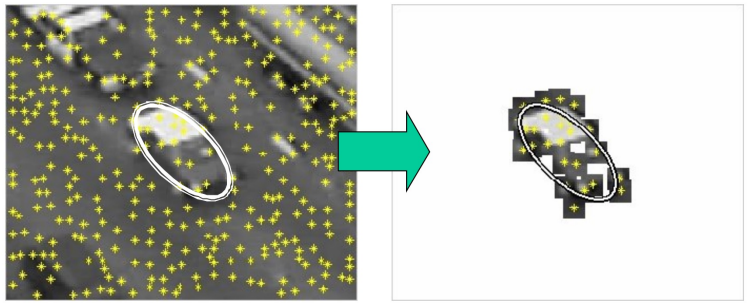


Image source -- https://www.cse.psu.edu/~rtc12/CSE486/lecture31_6pp.pdf

Instead of relying on manually crafted features, deep learning models can learn different representations from data automatically (i.e., the so-called deep features).

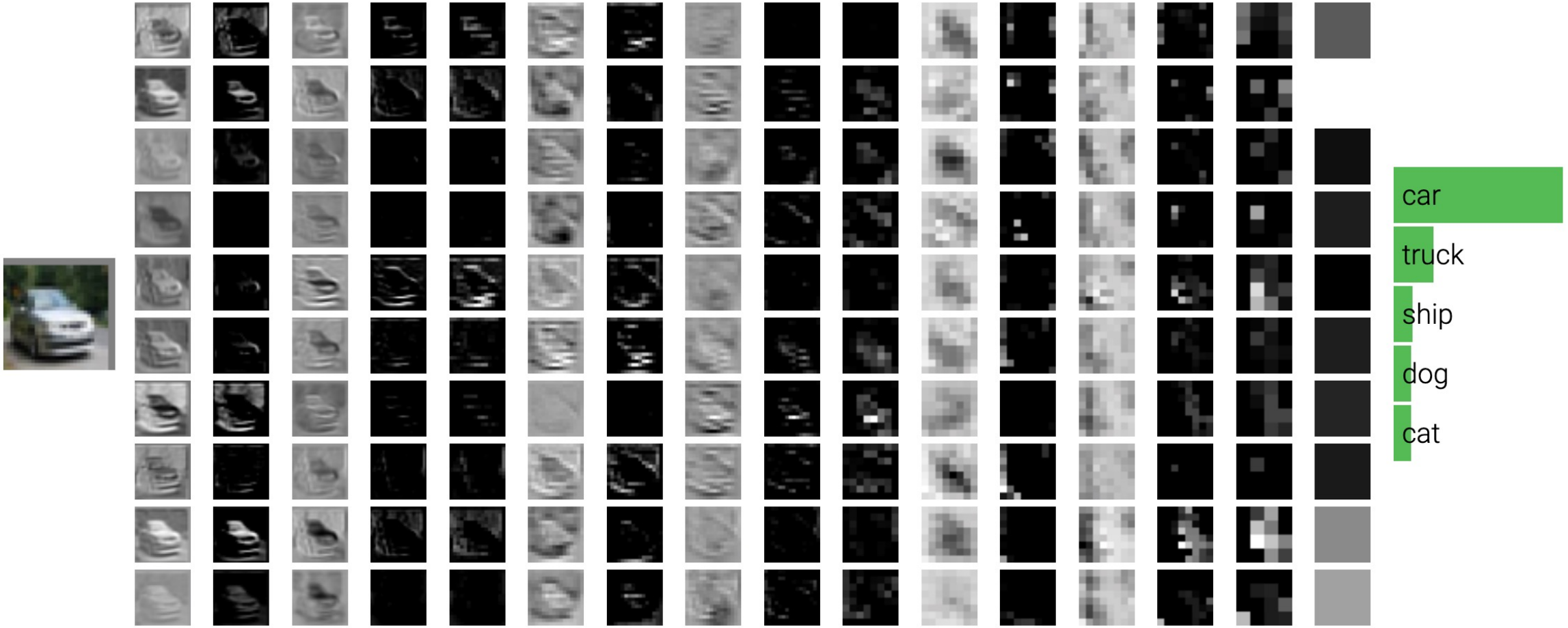


Image source -- http://cs231n.stanford.edu/

Deep learning models can extract features automatically and existed long ago but were not widespread due to the high demand for computational resources and power.

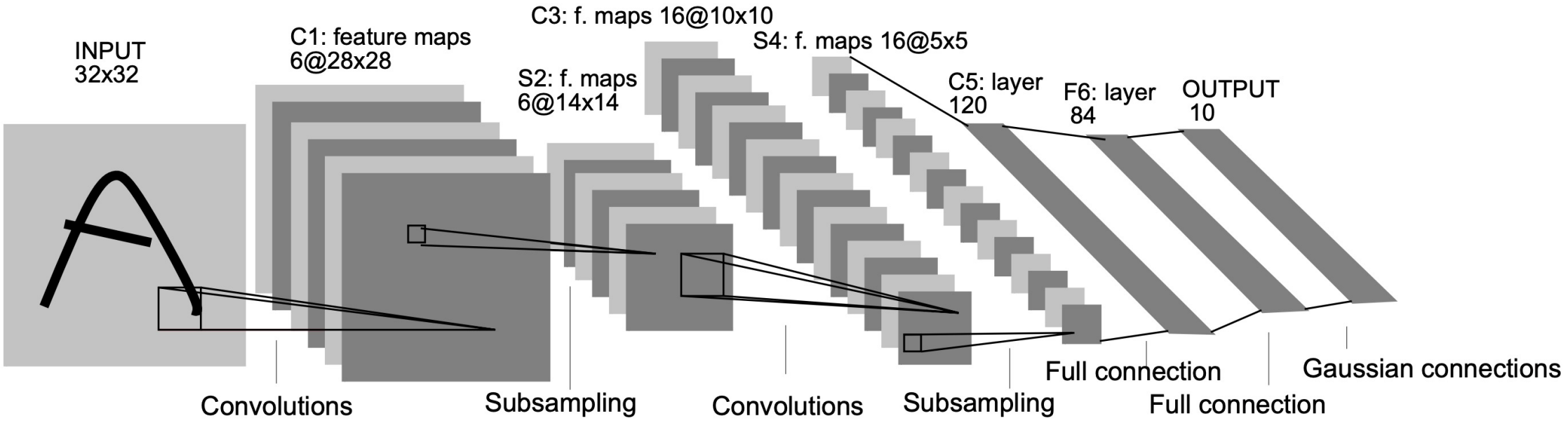


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeCun, Y., et al. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE.

In 2012, AlexNet showed great improvement in classifying images (ImageNet dataset) using a Convolutional Neural Network on GPUs. Then, the deep learning era started.

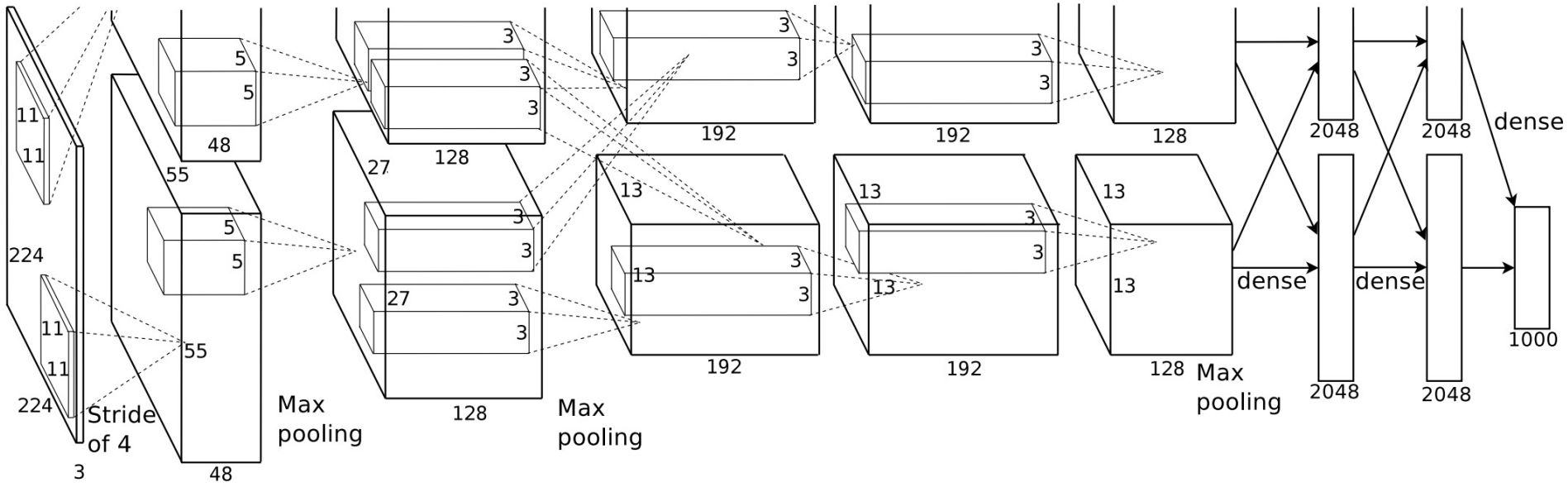
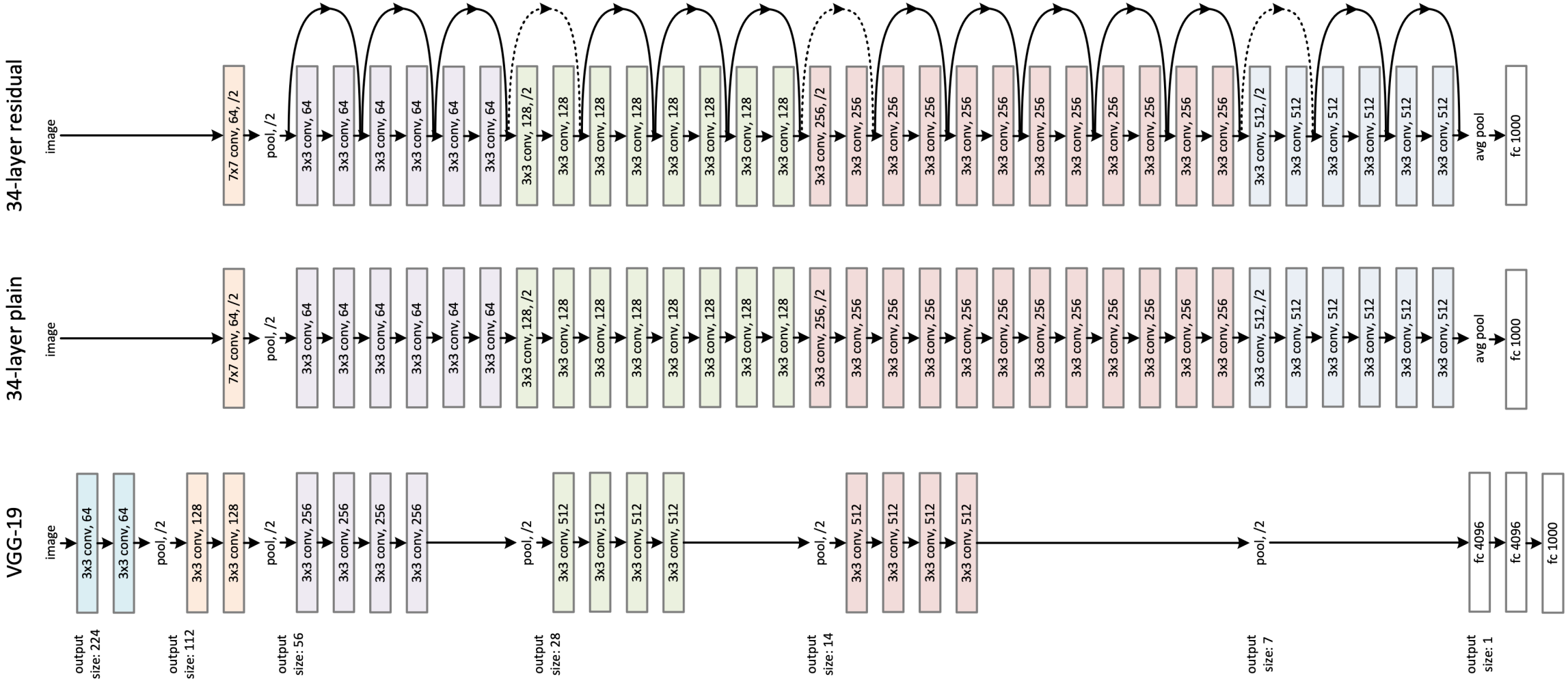


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Krizhevsky, A., et al. (2012). ImageNet classification with deep convolutional neural networks. Proceedings of the NeurIPS conference.

Then, deep neural networks went deeper and deeper with different variations.



He, K., et al. (2016). Deep residual learning for image recognition. Proceedings of the CVPR conference.

For sequential data, we can use the Recurrent Neural Network (RNN) architecture.

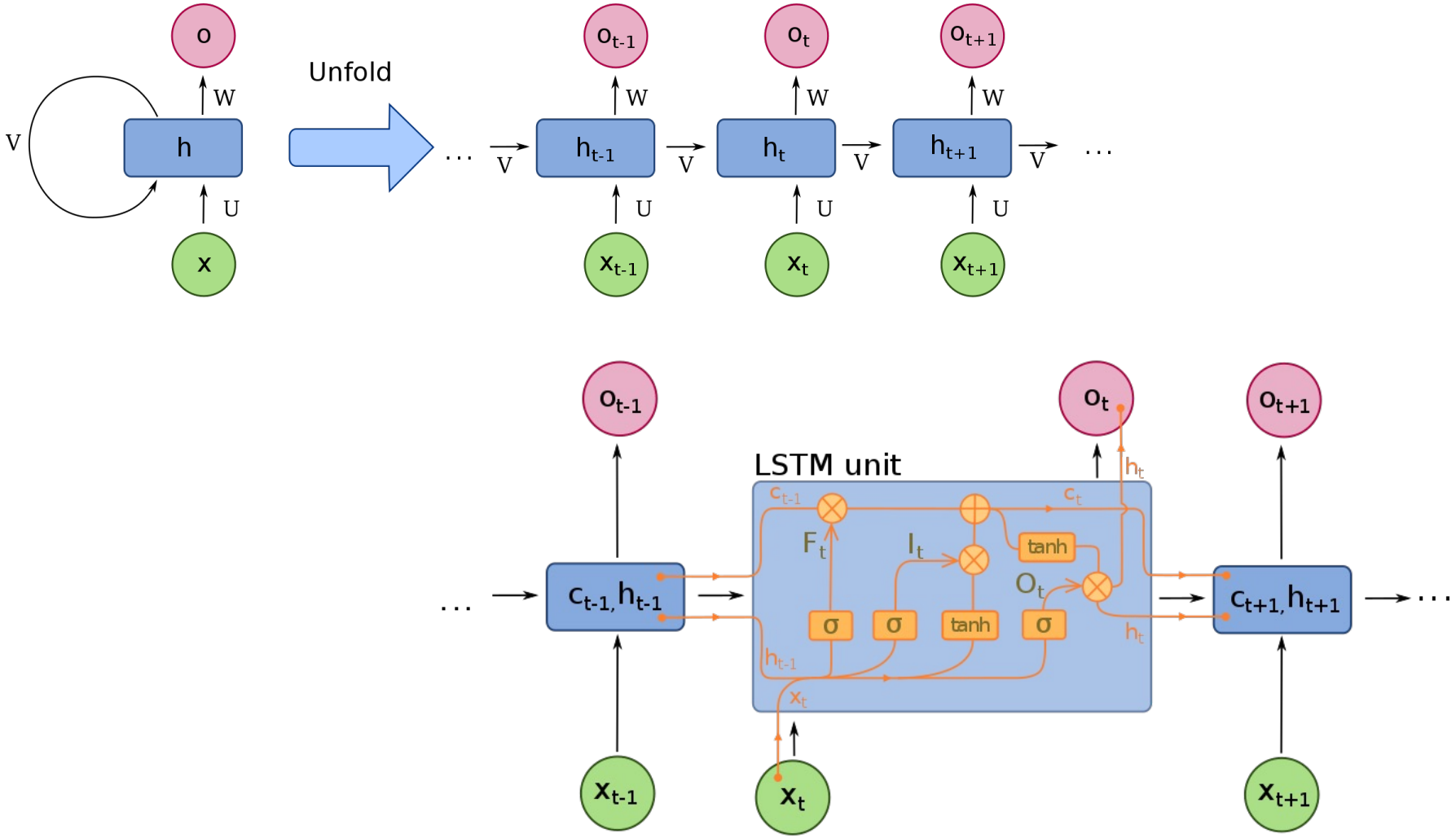
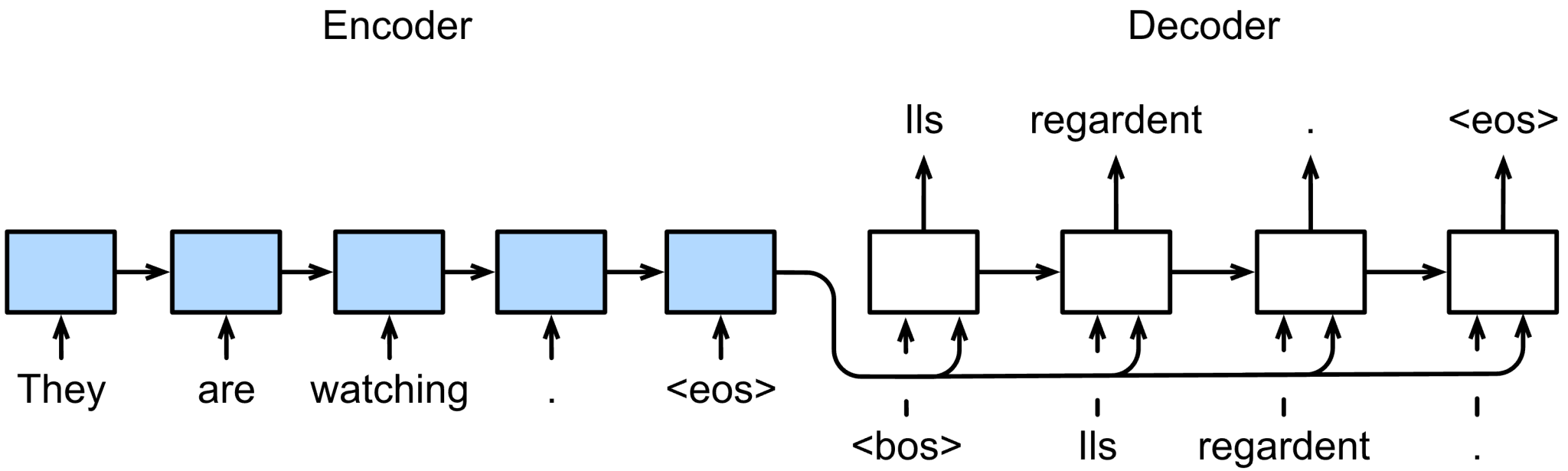


Figure source -- https://en.wikipedia.org/wiki/Recurrent_neural_network

For machine translation, the sequence-to-sequence model (which is an RNN) uses the encoder-decoder architecture. The encoder takes the input in one language, and the decoder outputs the translation to another language.



Source -- https://d2l.ai/chapter_recurrent-modern/seq2seq.html

We can use Autoencoder (based on combining encoder-decoder architecture) to perform image segmentation (using convolutional layers).

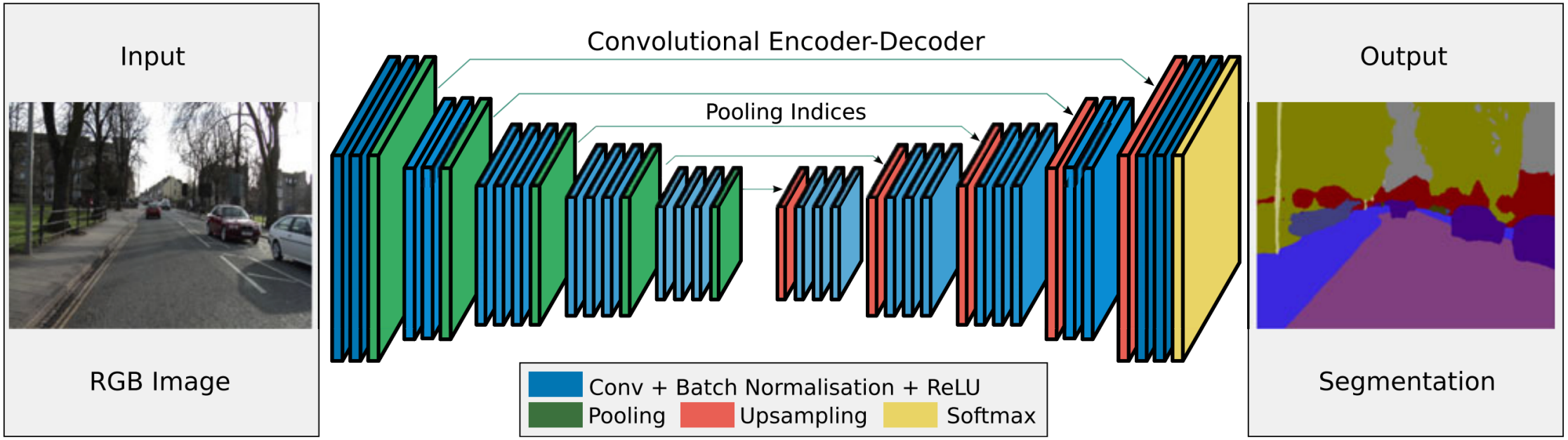


Fig. 2. An illustration of the SegNet architecture. There are no fully connected layers and hence it is only convolutional. A decoder upsamples its input using the transferred pool indices from its encoder to produce a sparse feature map(s). It then performs convolution with a trainable filter bank to densify the feature map. The final decoder output feature maps are fed to a soft-max classifier for pixel-wise classification.

Badrinarayanan, V., et al. (2017). SegNet: A deep convolutional encoder-decoder architecture for image segmentation. IEEE.

Many current works are based on the Transformer architecture (also encoder-decoder).

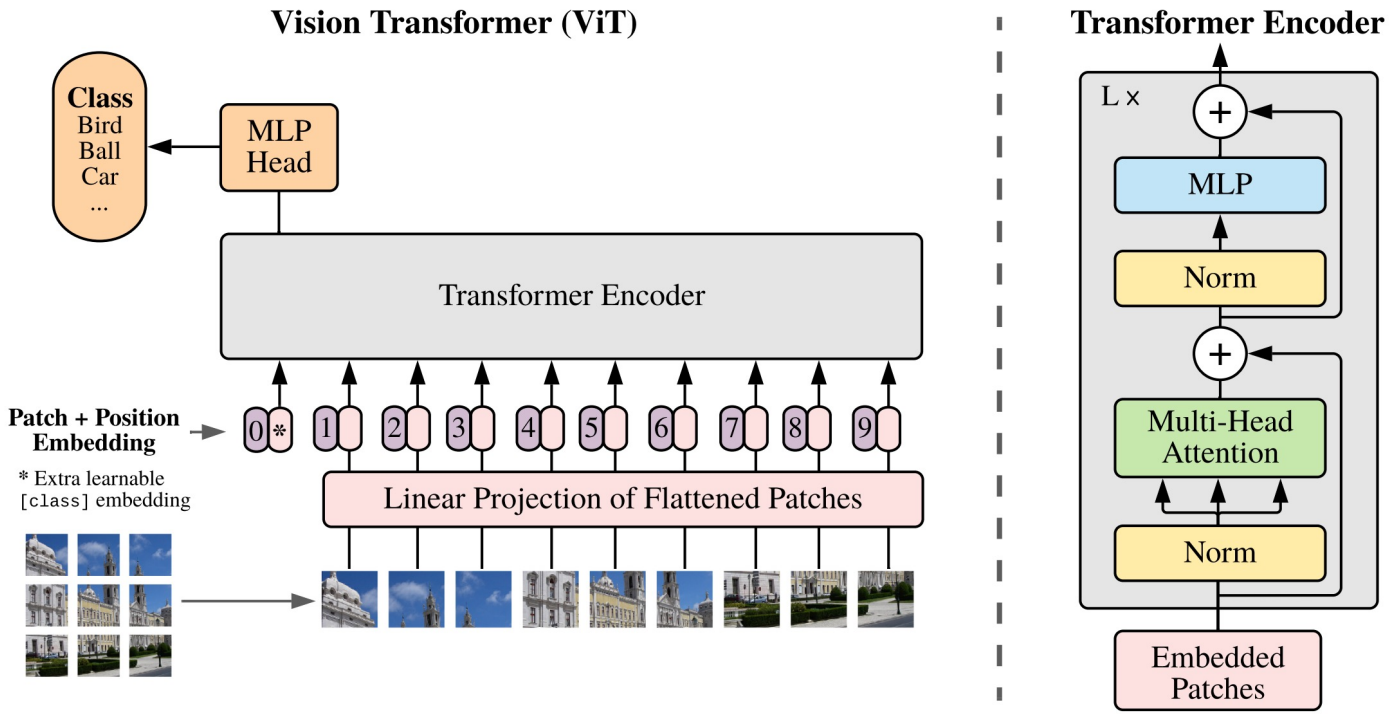
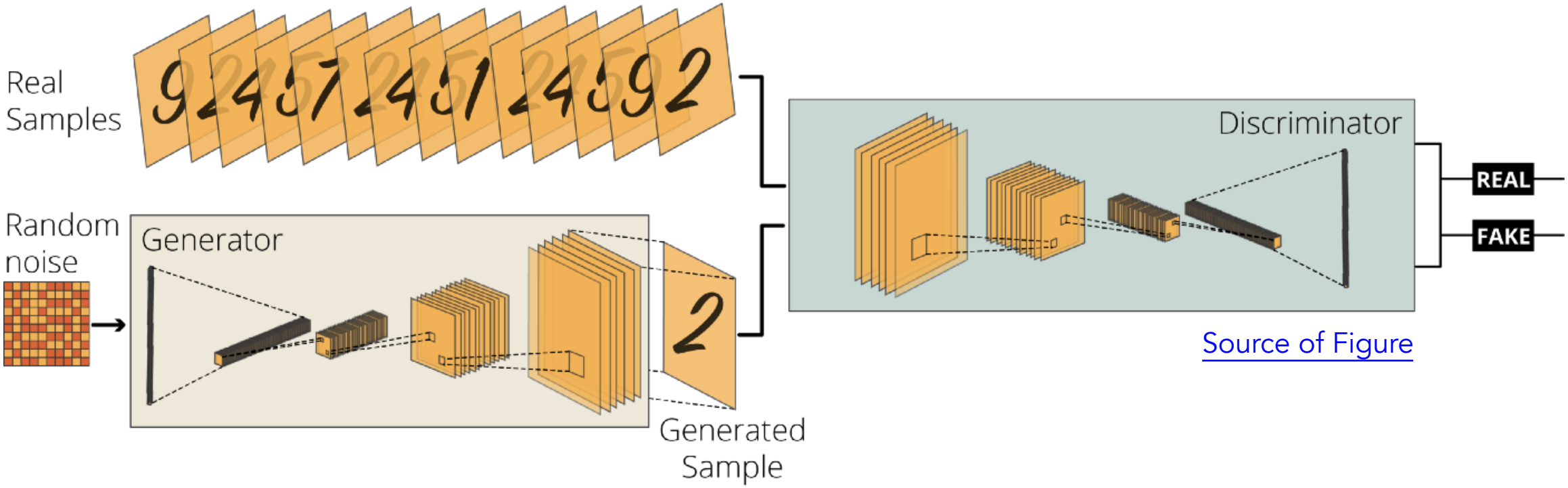


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

Dosovitskiy, A., et al. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In the ICLR conference.

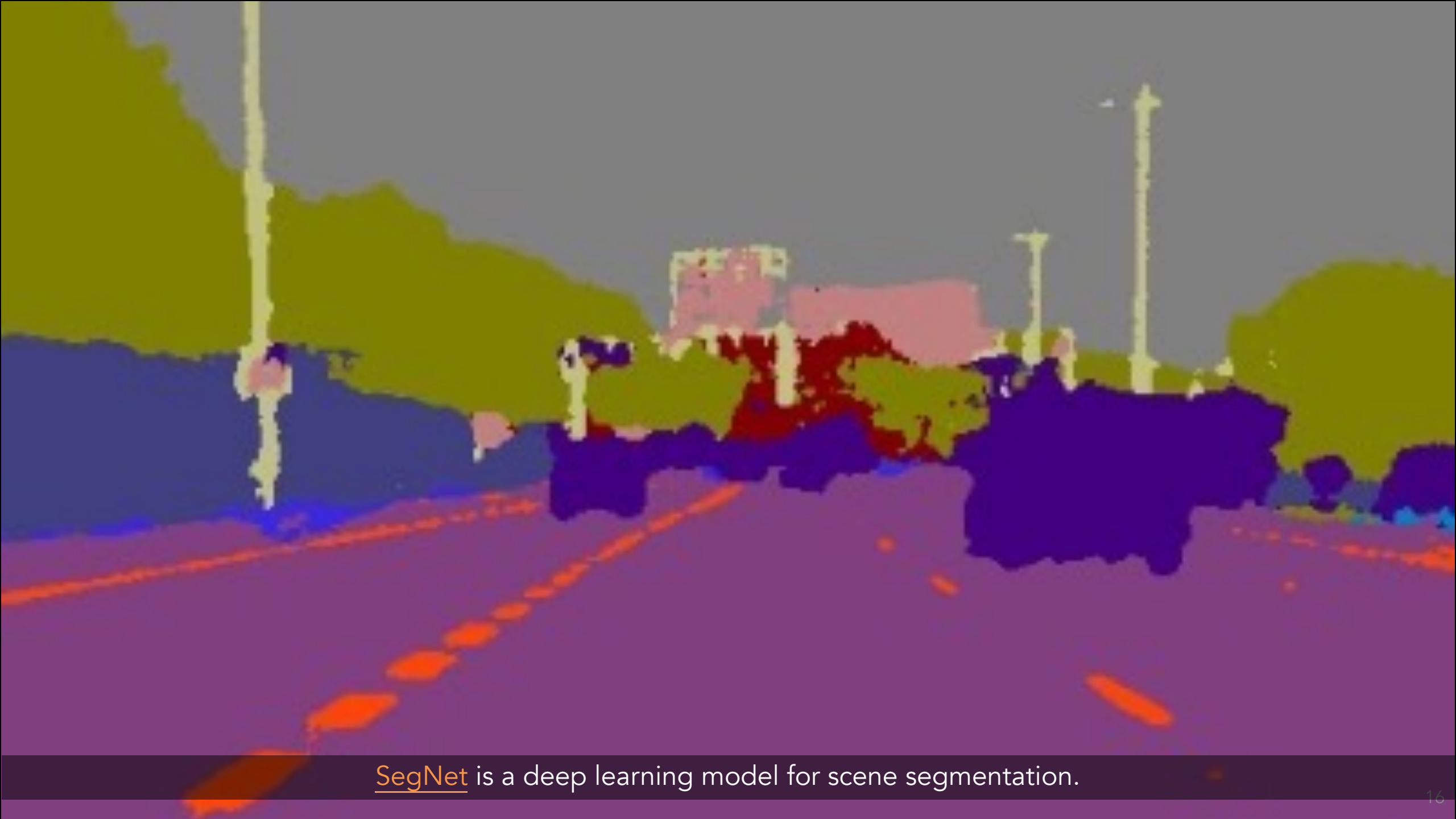
We can also use deep learning to generate data, such as the Generative Adversarial Network structure, which combines a generator (that converts noise to a fake sample) and a discriminator (that tries to identify if a sample is fake or real).



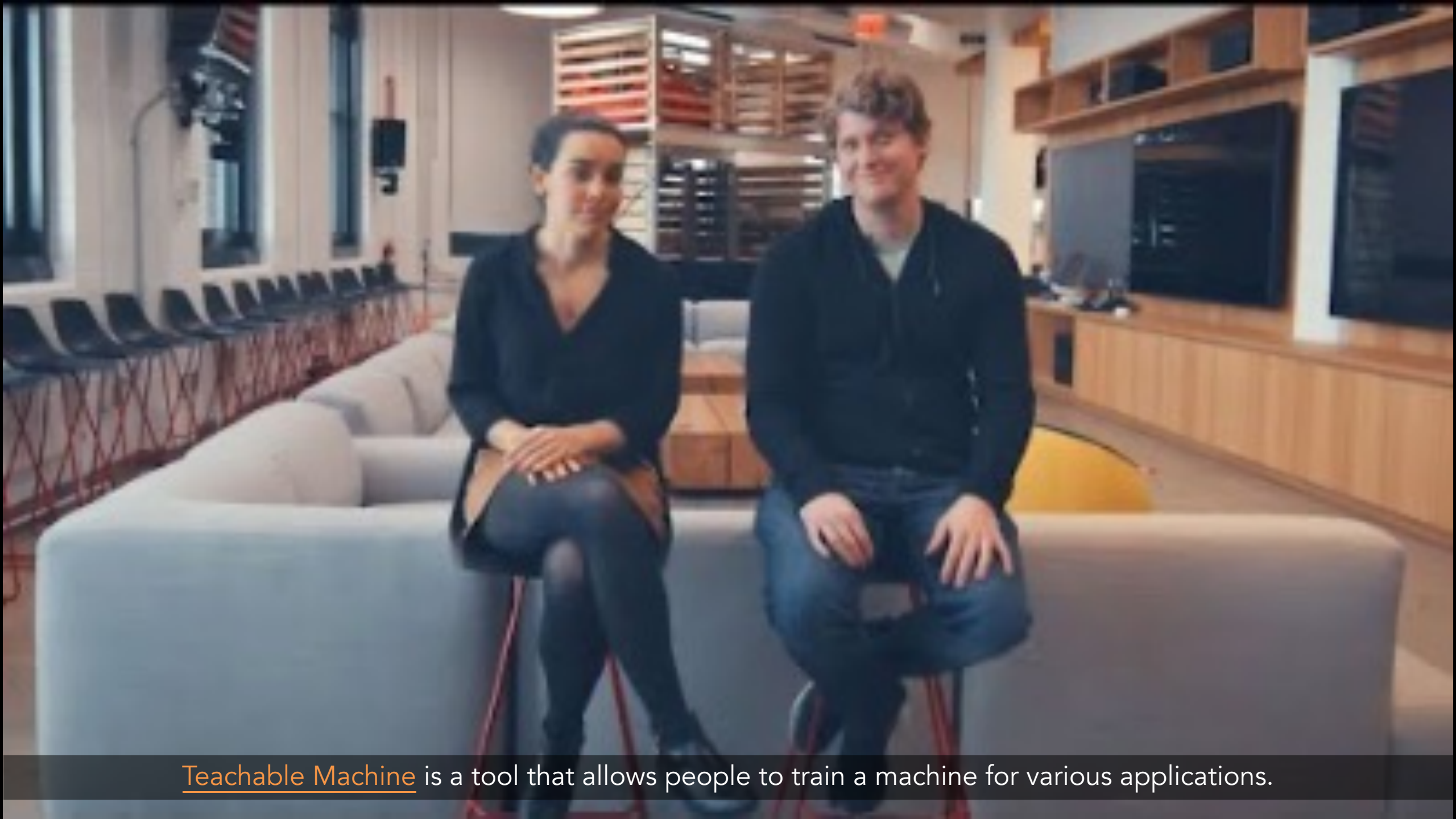
Source of Figure

More about the Generative Adversarial Network -- https://sthalles.github.io/intro-to-gans/

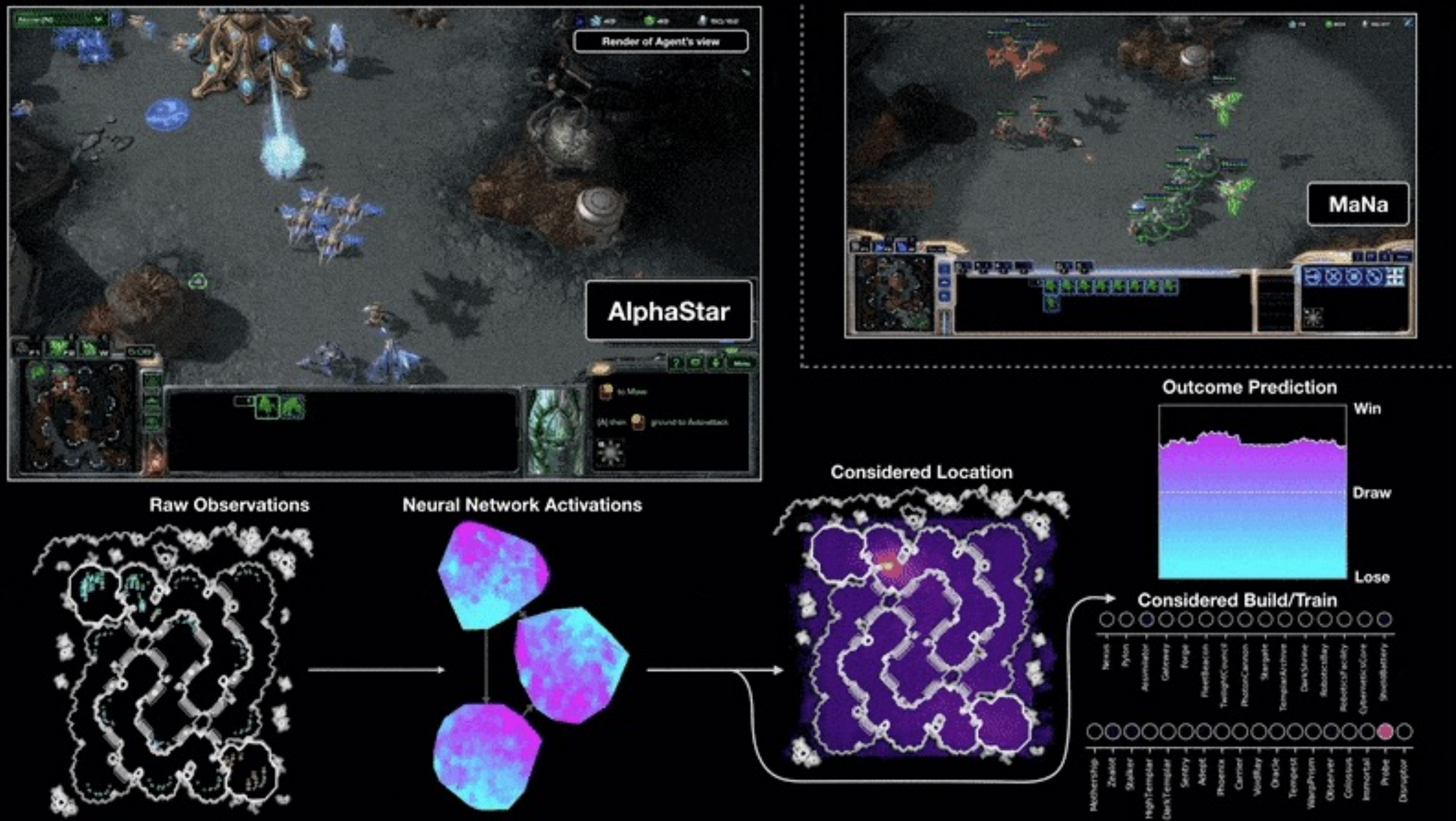
YOLO is a deep learning model for object detection.



SegNet is a deep learning model for scene segmentation.

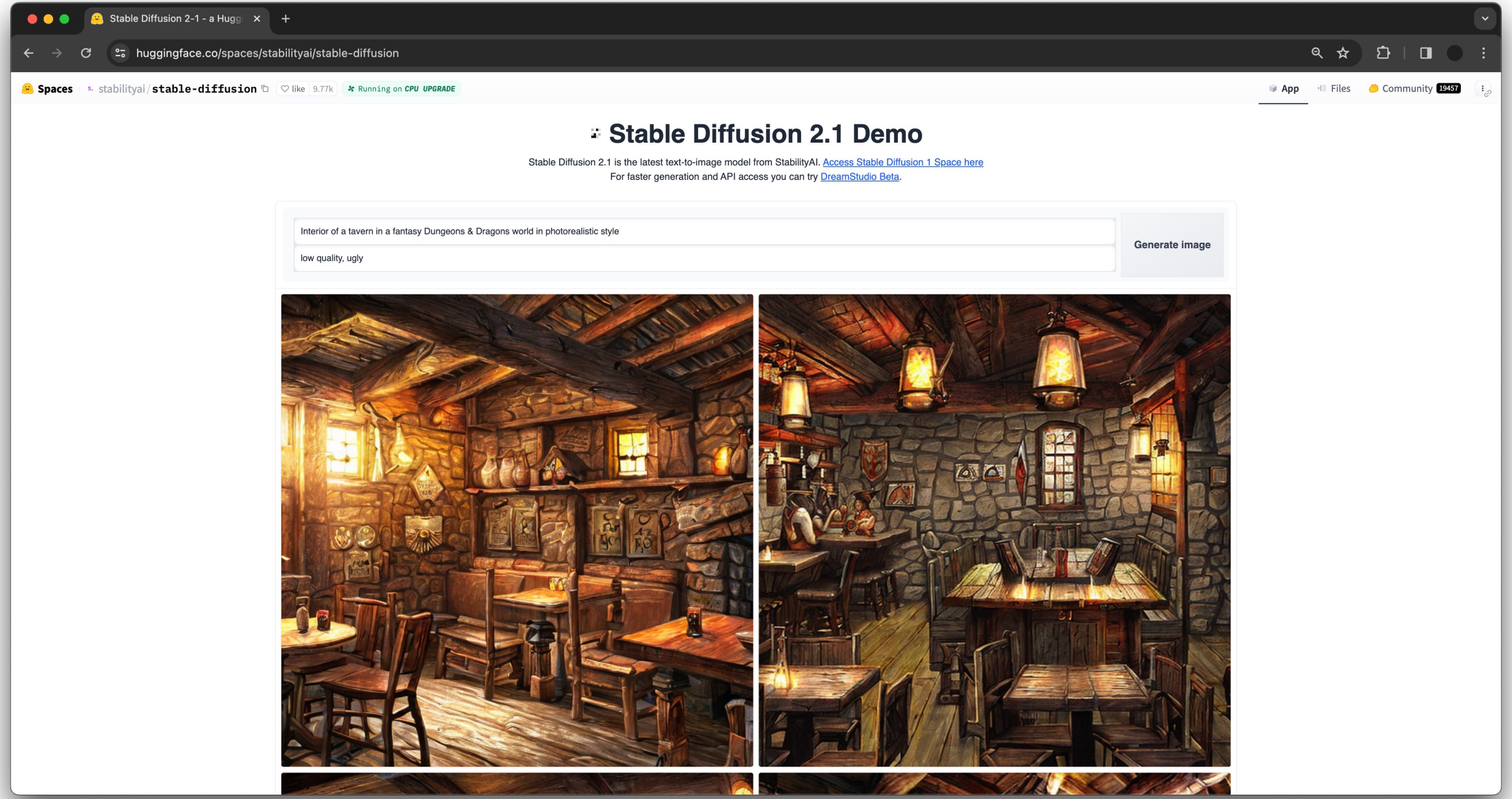

Teachable Machine is a tool that allows people to train a machine for various applications.

https://www.deepmind.com/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii

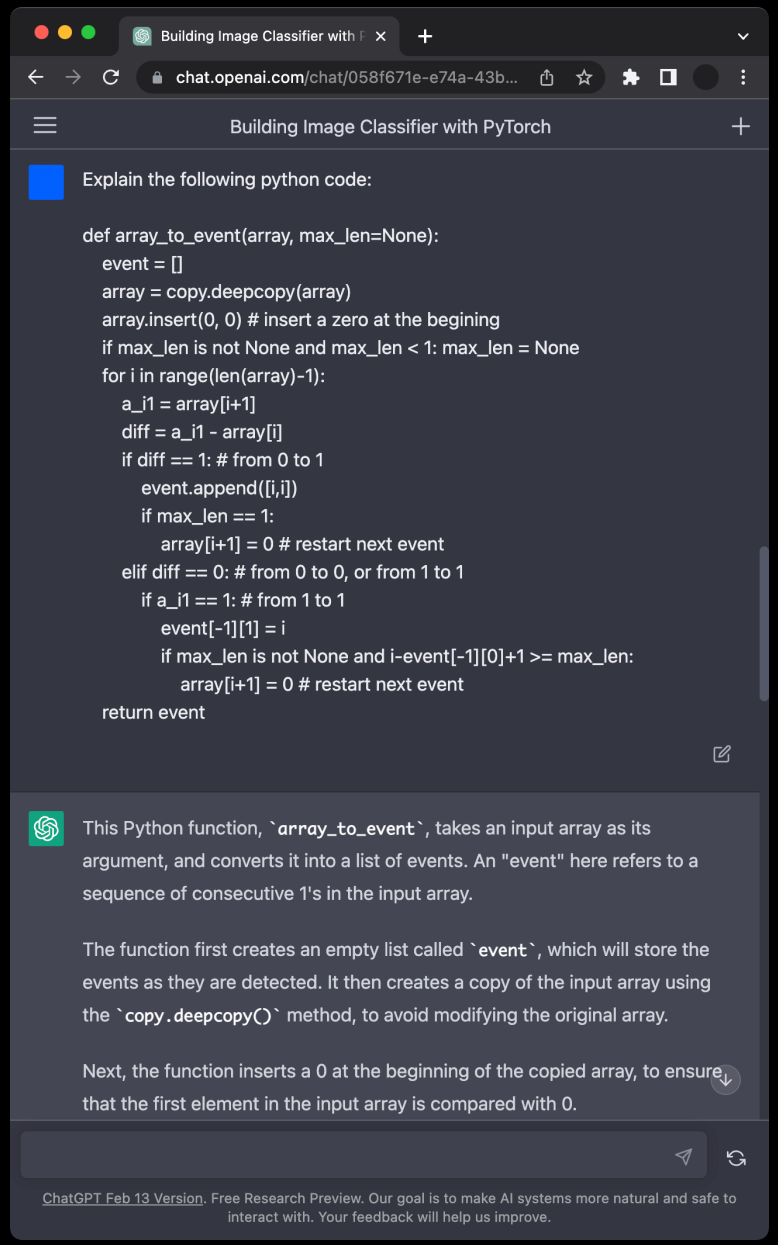


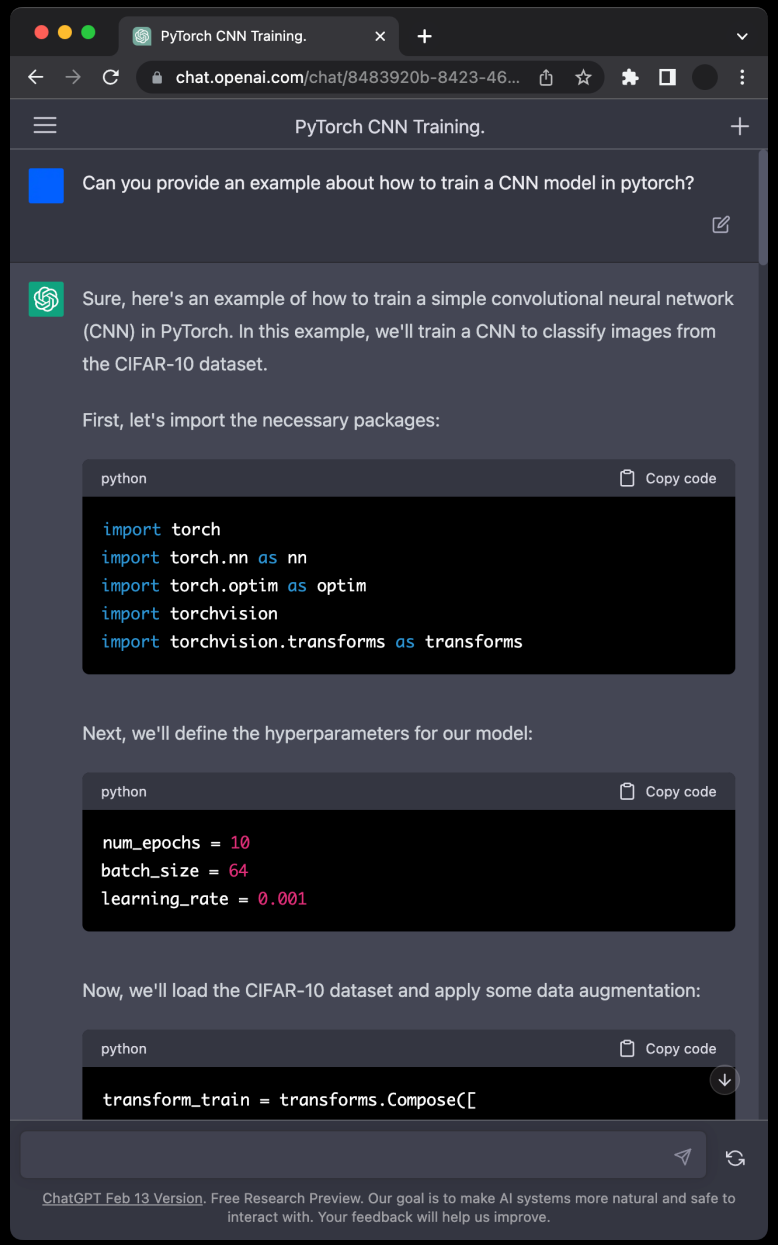
This link has a short highlight about how AlphaStar plays StarCraft II.

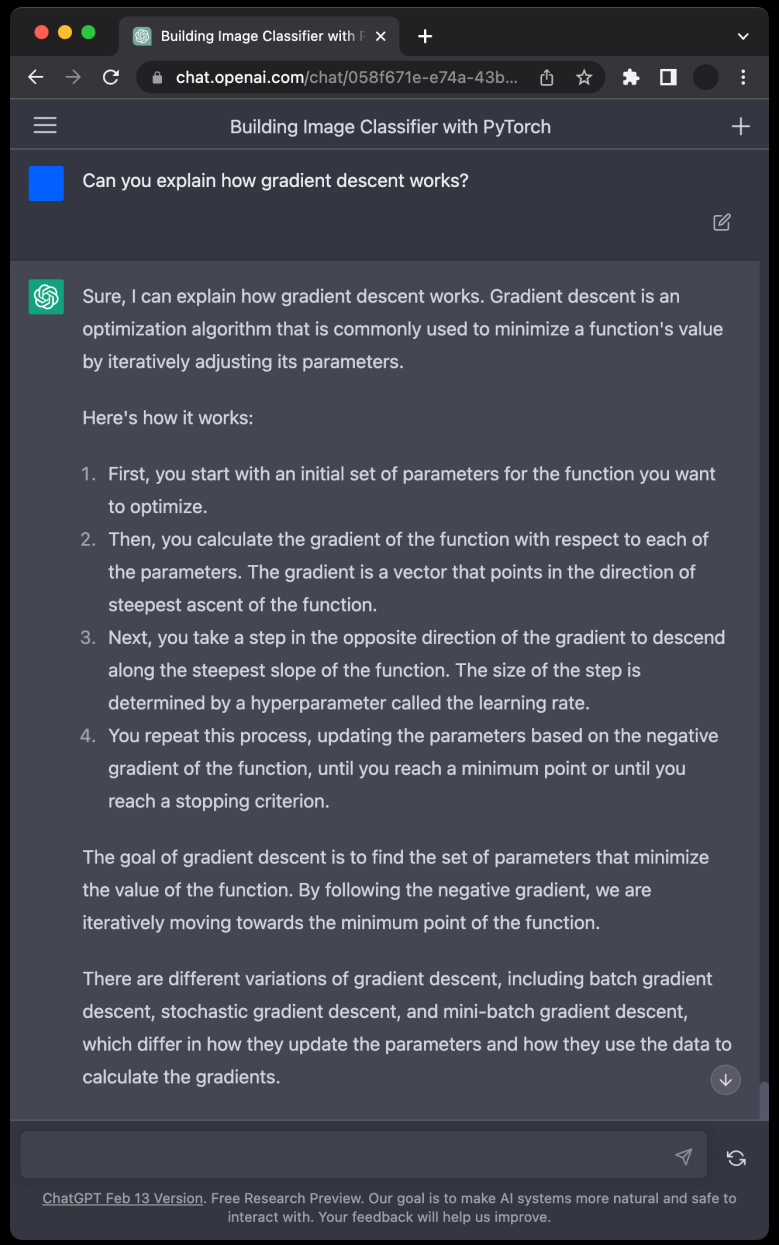
DALL·E 2 uses deep learning to create realistic images and art from text descriptions.



Stable Diffusion -- https://huggingface.co/spaces/stabilityai/stable-diffusion



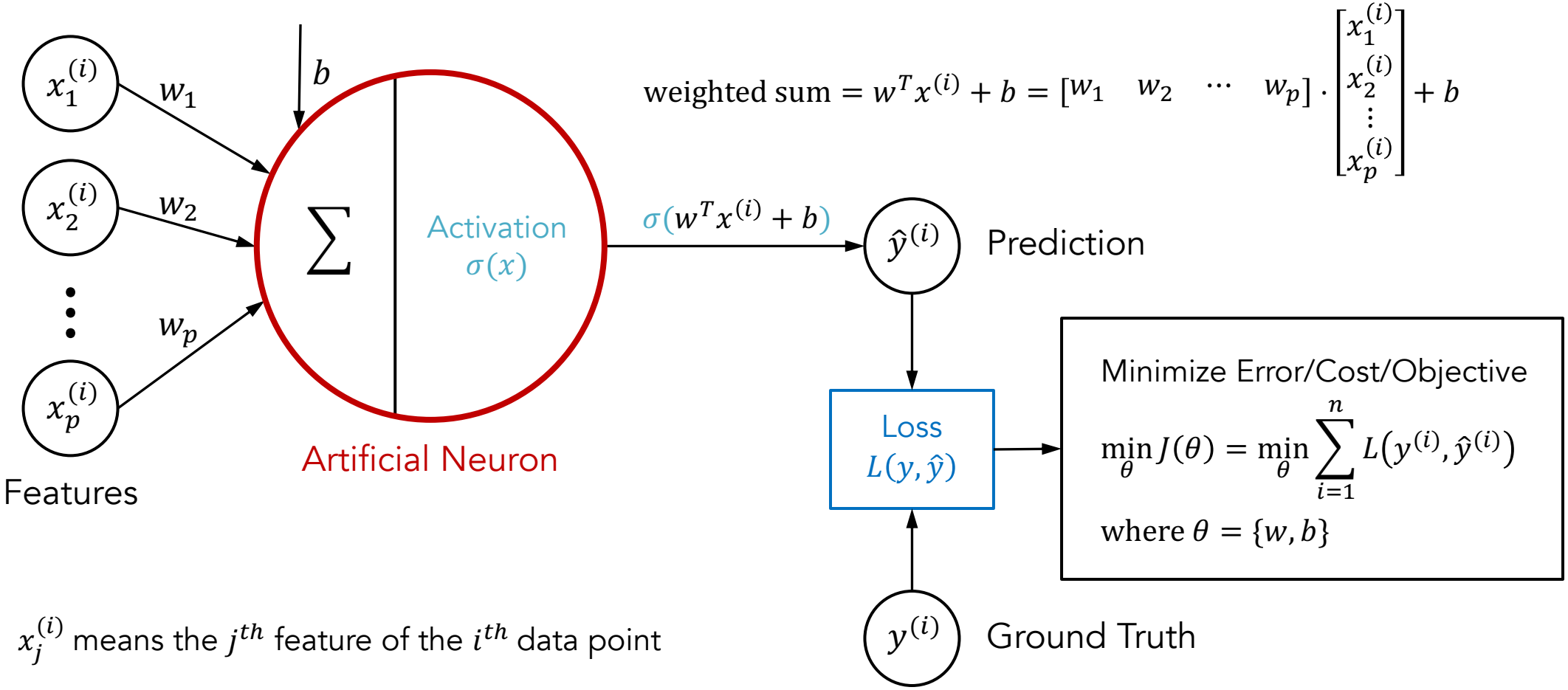




ChatGPT is a large language model developed by OpenAI to have natural conversations with people.

Now let us look at how a deep neural net works, starting from the artificial neuron.

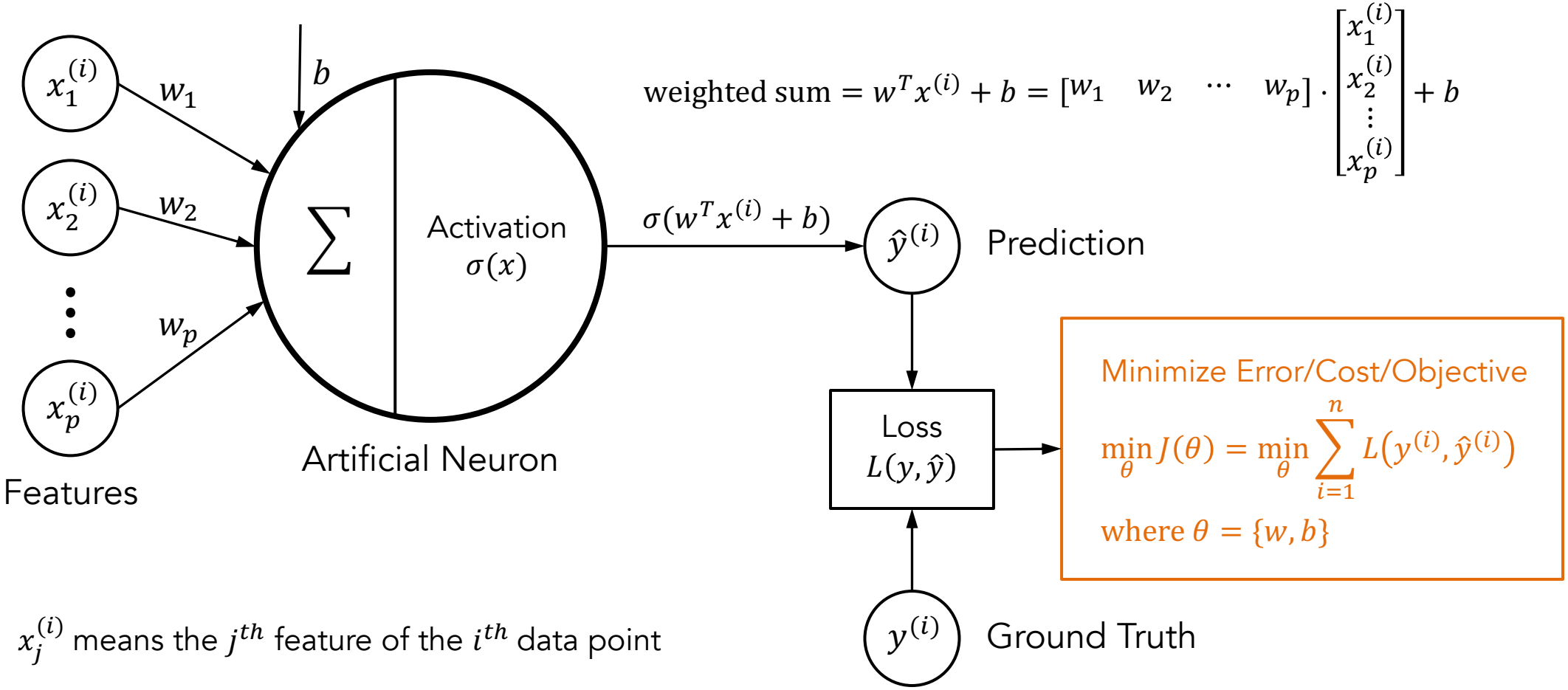
An artificial neuron can convert feature x to prediction $\hat{y}$ by using a weighted sum and an activation function. Then, we need to define a loss function based on the task type.



$x_j^{(i)}$ means the j^{th} feature of the i^{th} data point

We need to minimize the error/cost to find the optimal parameter set $\theta = \{w, b\}$.

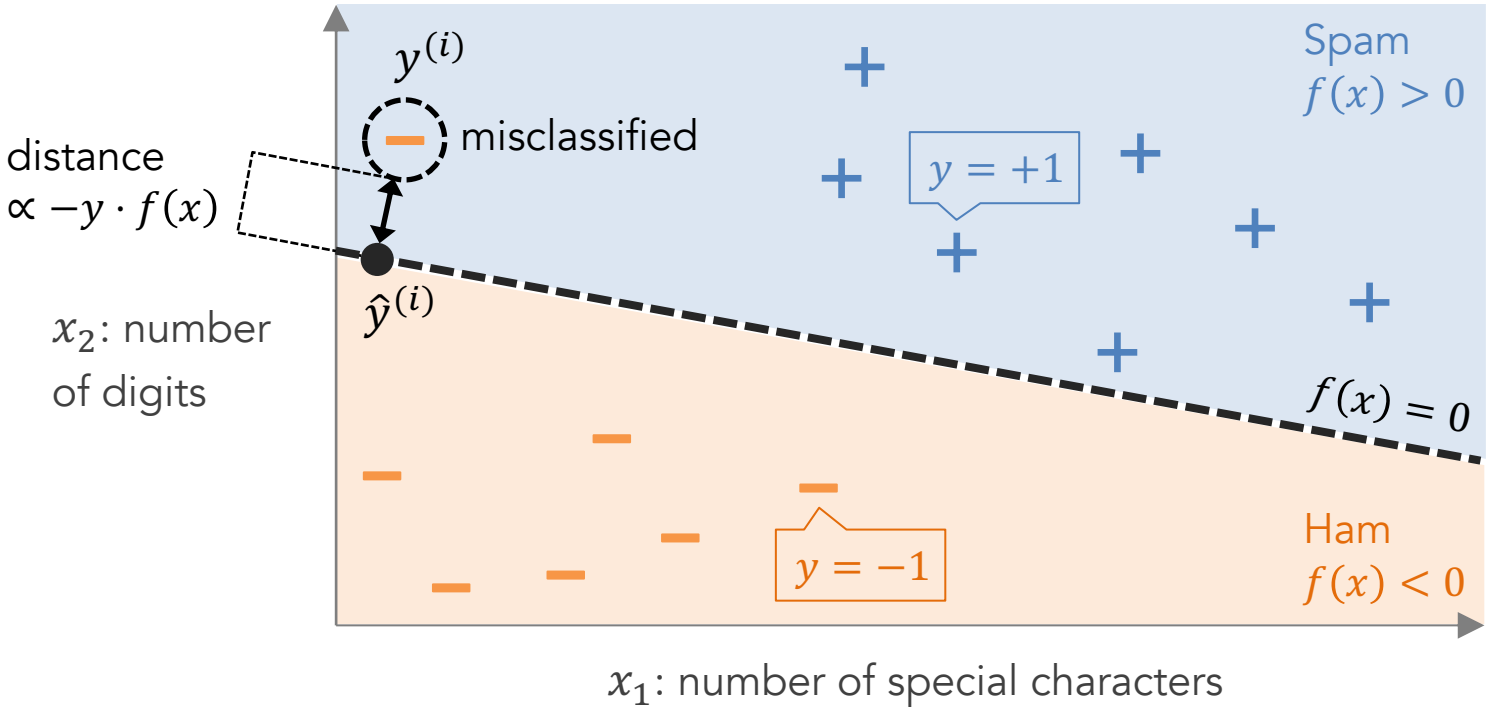
Notice that the prediction $\hat{y}$ is a function of θ, meaning $\hat{y}(\theta) = \hat{y}(w, b) = \sigma(w^T x + b)$.

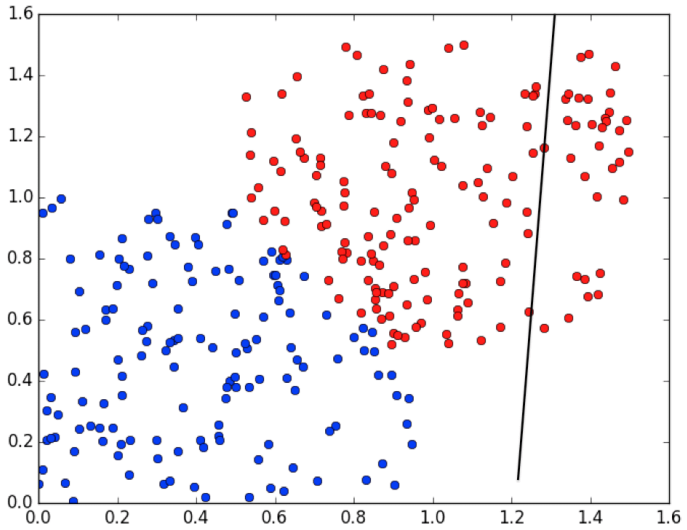


$x_j^{(i)}$ means the j^{th} feature of the i^{th} data point

We have learned how to classify spam and ham using the perceptron algorithm with an error metric. We can also formulate the error in another way.

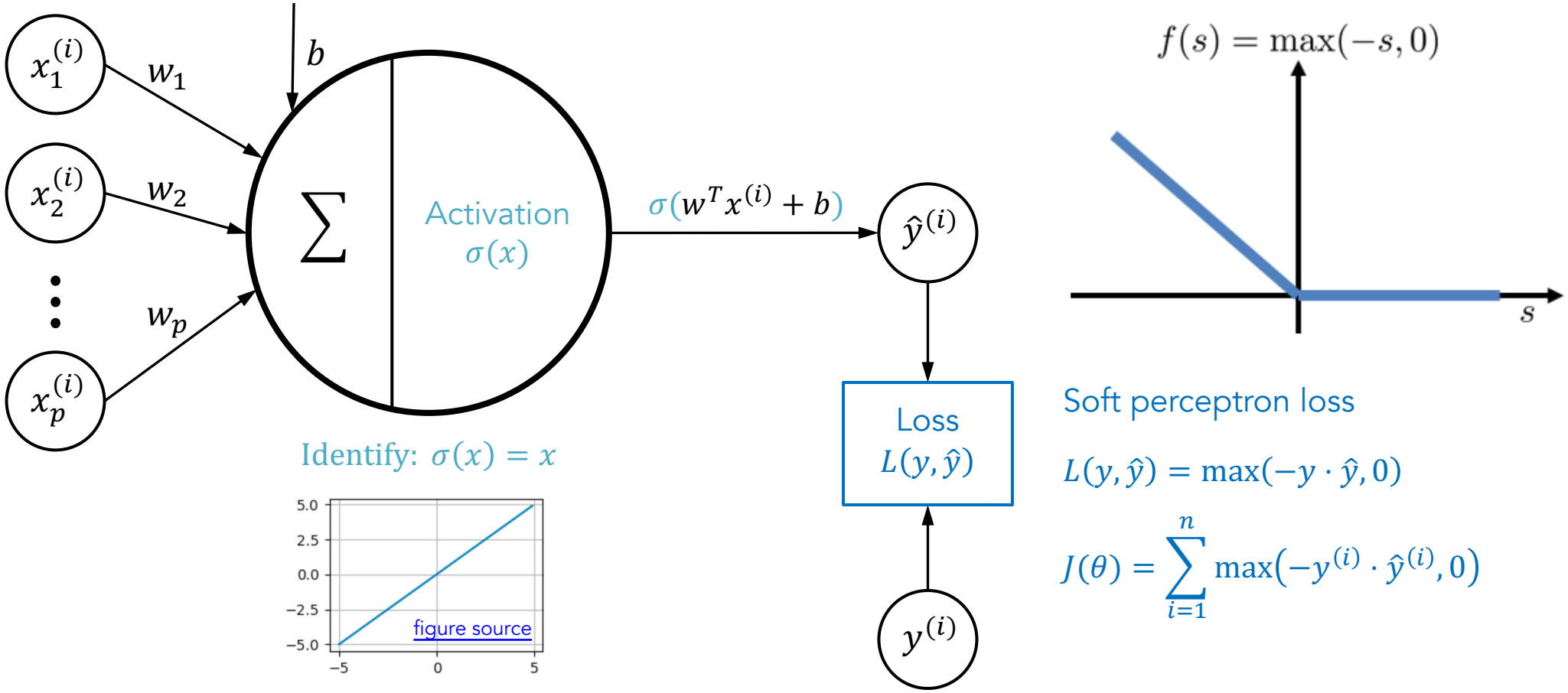
$$\text{error} = \sum_{i=1}^{n} -y^{(i)} \cdot f(x^{(i)}) \text{ for each misclassified point} = \sum_{i=1}^{n} \max(-y^{(i)} \cdot \hat{y}^{(i)}, 0) \text{ for all points}$$





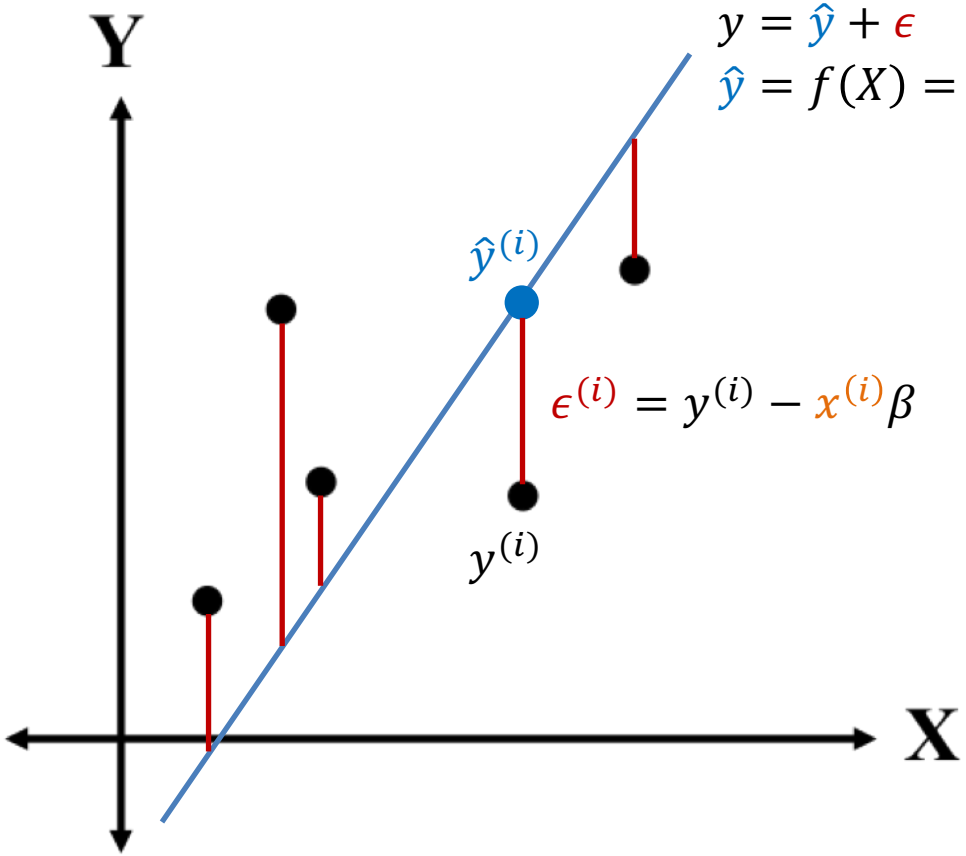
Rosenblatt's Perceptron Learning Algorithm: section 4.5.1 in book https://hastie.su.domains/ElemStatLearn/

We can represent the perceptron classifier using an artificial neuron. In this case, we use the identify function (as the activation) with the soft perceptron loss.



More information about perceptron in a neural net -- https://engineering.purdue.edu/ChanGroup/ECE595/files/Lecture16_perceptron1.pdf

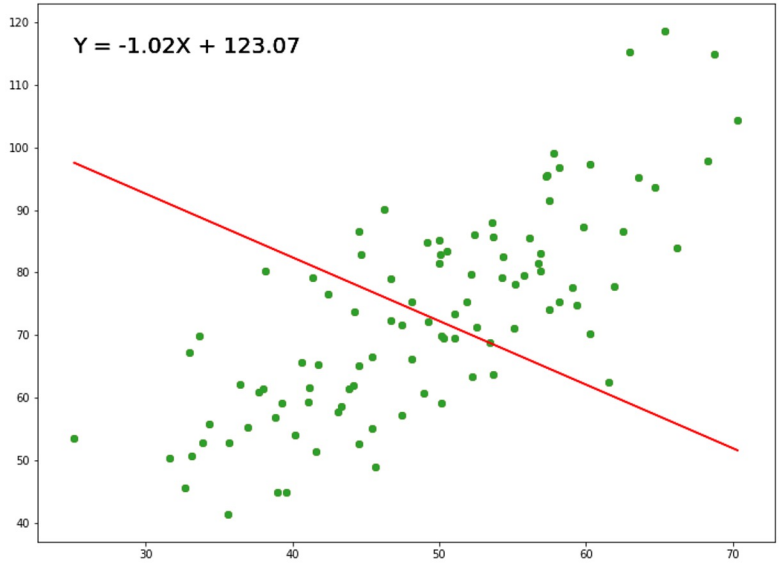
We have also learned linear regression, where we need to find a set of coefficients that can minimize the mean squared errors between all the predictions and ground truths.



$$y = \hat{y} + \epsilon$$
$$\hat{y} = f(X) = X\beta$$

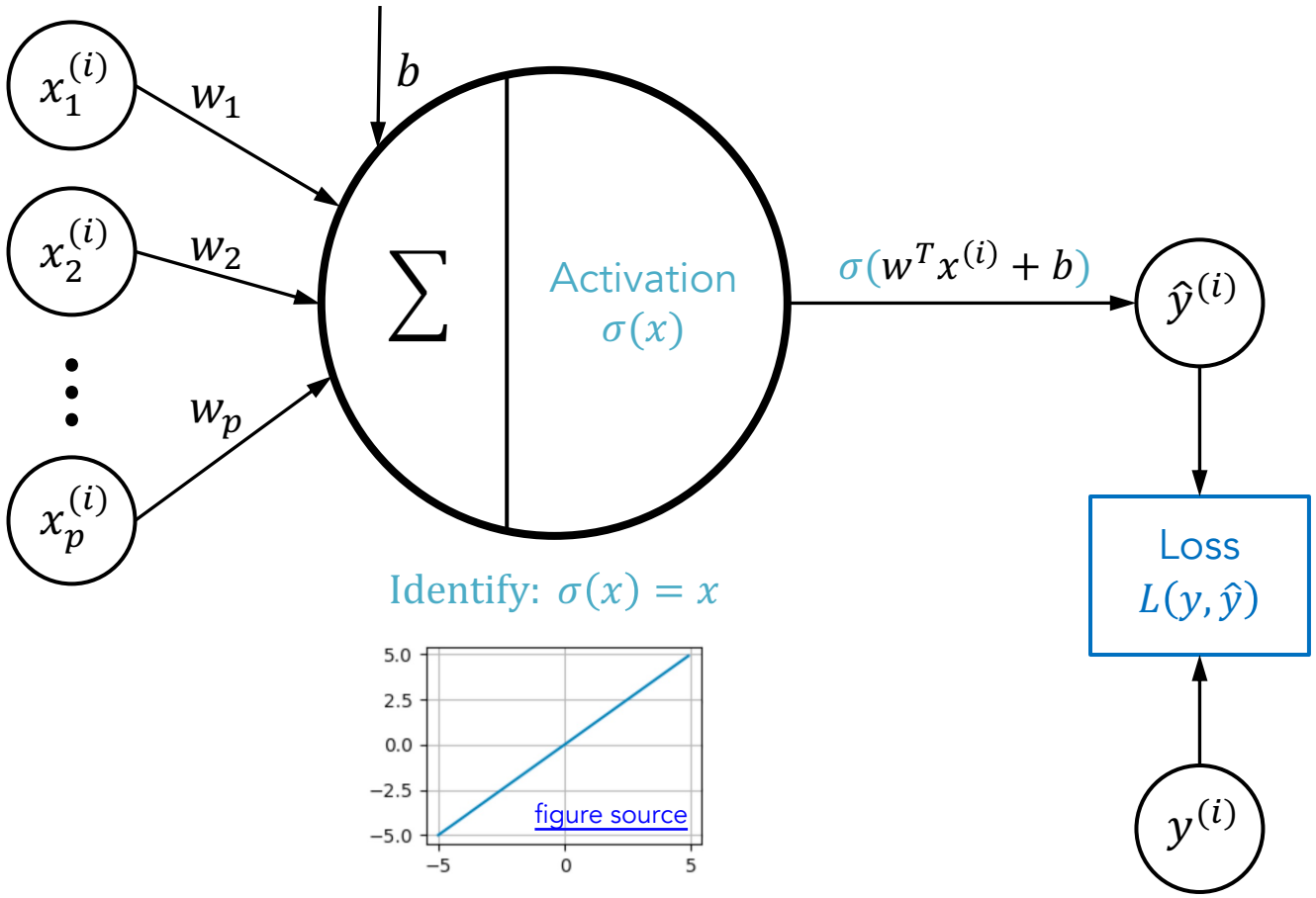
$$\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} \left(\epsilon^{(i)}\right)^2 = \min_{\beta} \frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} - x^{(i)}\beta\right)^2$$

$$= \min_{\beta} \frac{1}{n} (y - X\beta)^T (y - X\beta)$$

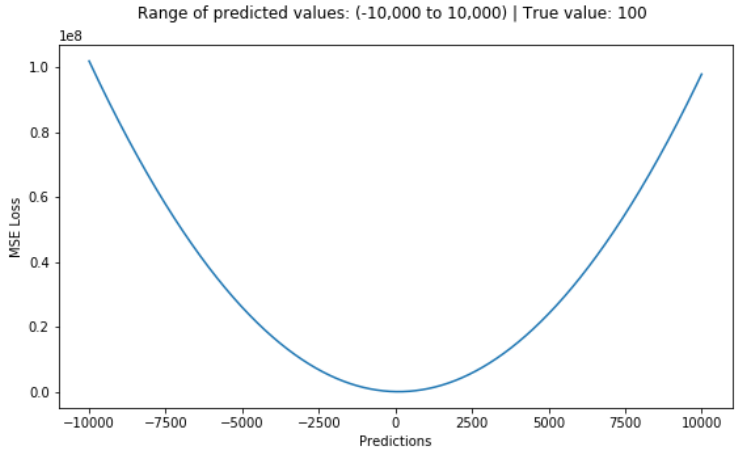


More about linear regression -- https://www.stat.cmu.edu/~cshalizi/mreg/15/lectures/13/lecture-13.pdf

We can also represent the linear regression model using an artificial neuron. In this case, we use the identity activation function with the squared error loss function.



Identify: $\sigma(x) = x$



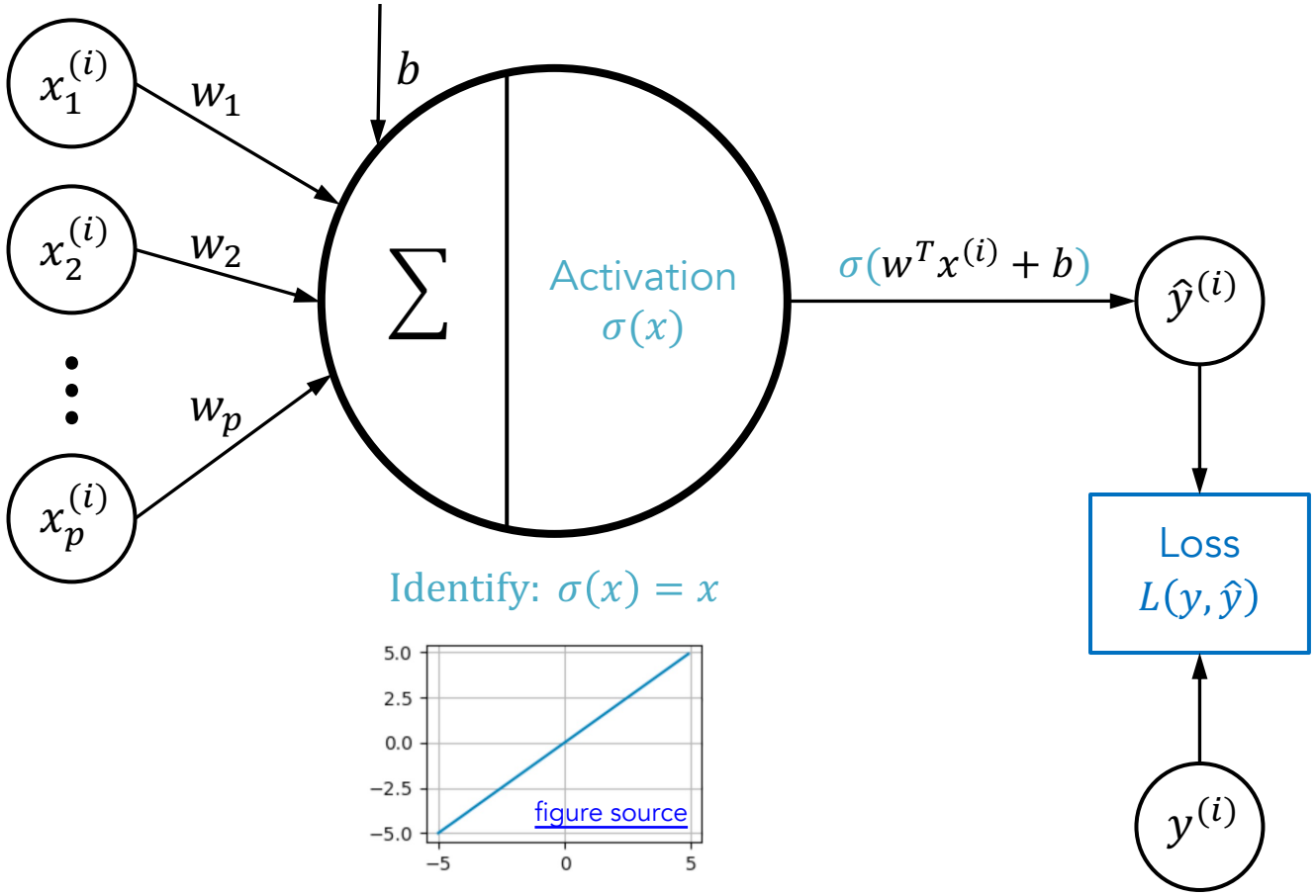
Squared error loss

$$L(y, \hat{y}) = (y - \hat{y})^2$$

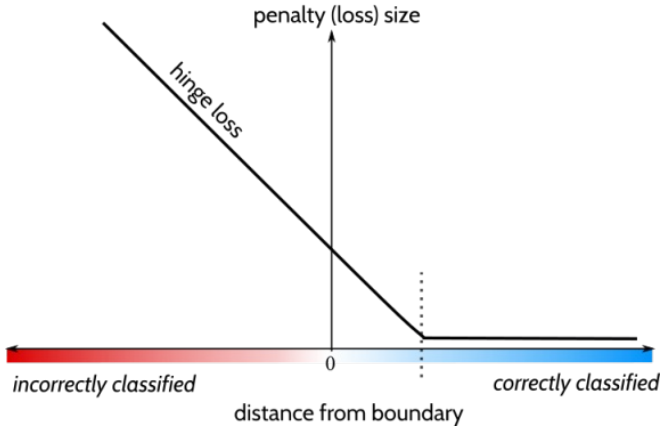
$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$$

MSE loss -- https://heartbeat.comet.ml/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0

We can replace the activation and loss functions with different ones to build another model. The example below uses hinge loss, which becomes Support Vector Machine.



Identify: $\sigma(x) = x$



Hinge loss

$$L(y, \hat{y}) = \max(1 - y \cdot \hat{y}, 0)$$

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \max\left(1 - y^{(i)} \cdot \hat{y}^{(i)}, 0\right)$$

Hinge loss -- https://math.stackexchange.com/questions/782586/how-do-you-minimize-hinge-loss

Support Vector Machine (right figure) finds the maximum margin γ separating hyperplane $h(x)$ for classification, while the perceptron classifier finds a separating hyperplane if it exists (e.g., blue or red line on the left figure).

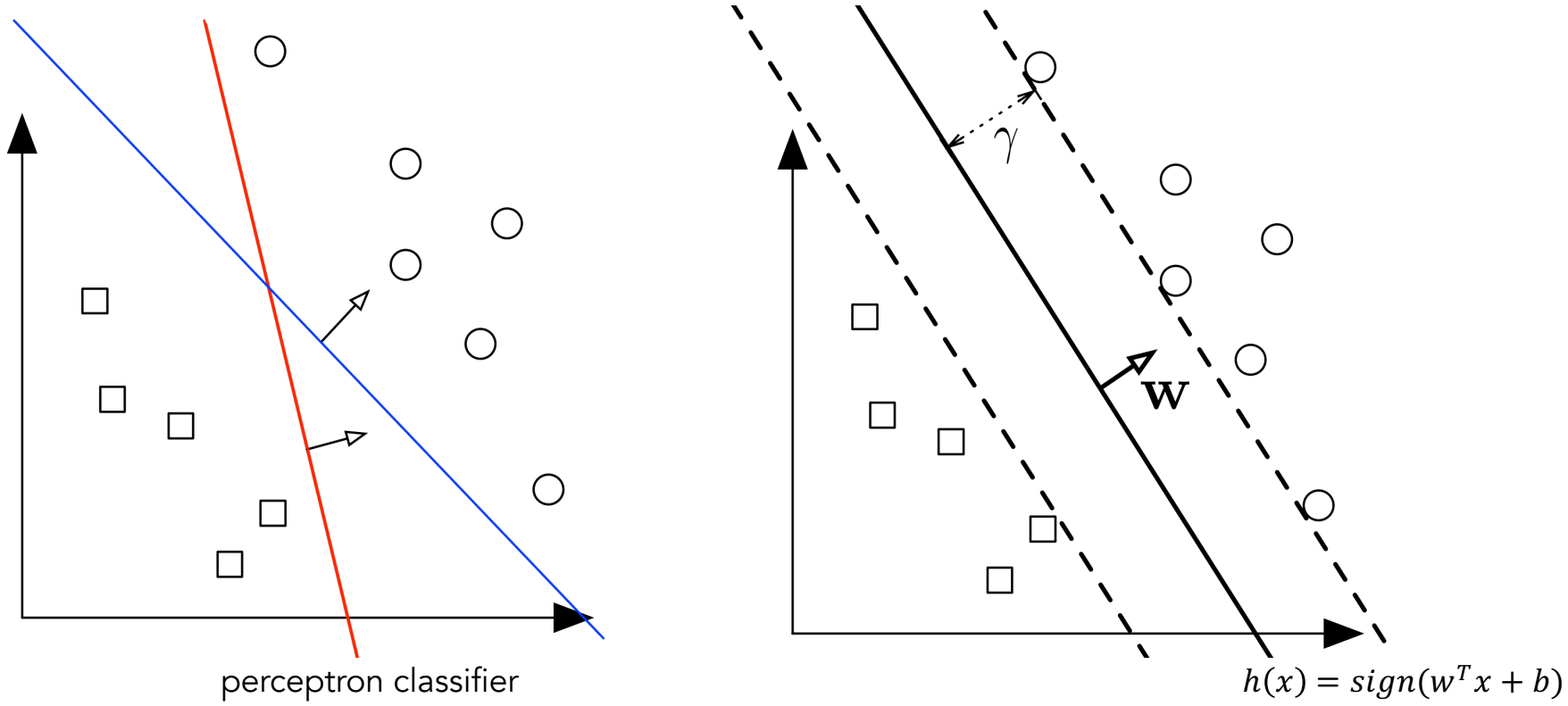
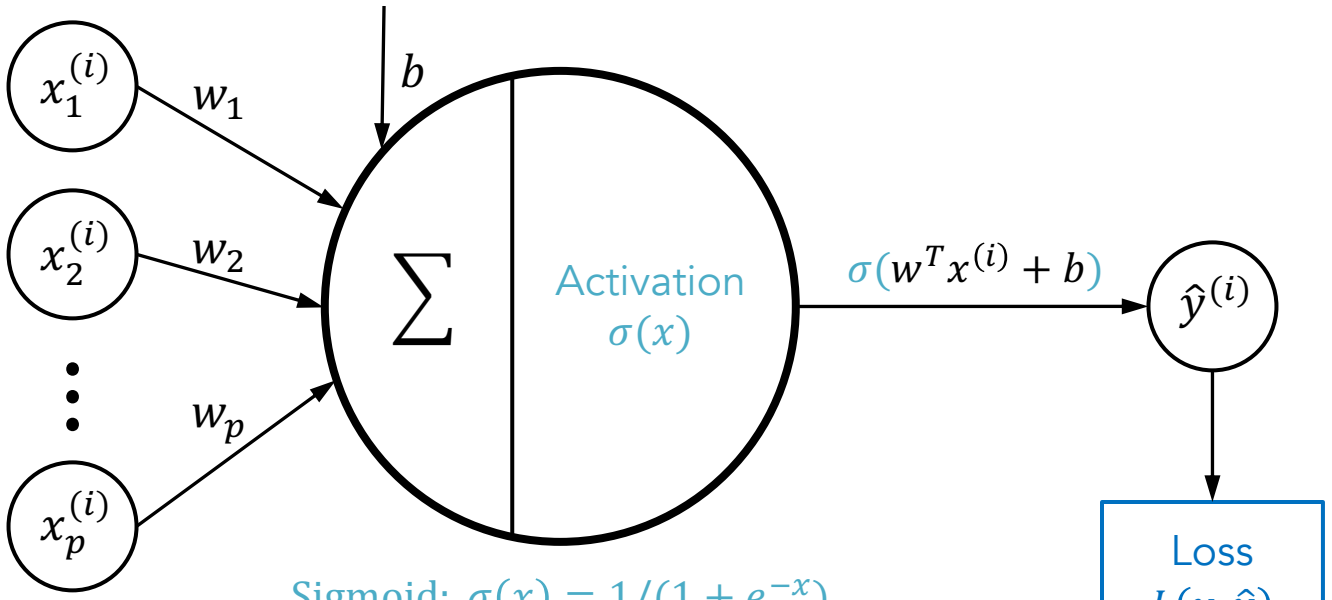
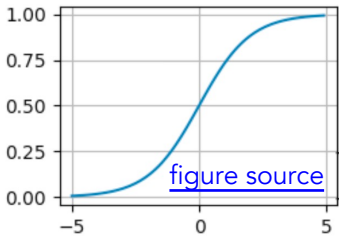


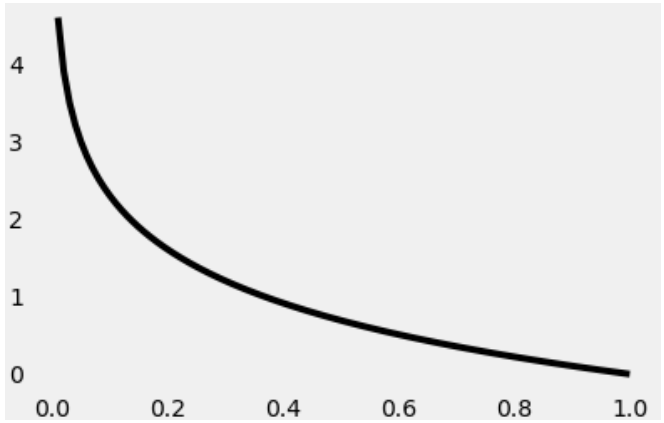
Figure source -- https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote09.html

If we replace the activation function to sigmoid and use the logistic loss (i.e., the binary version of cross-entropy loss), the neuron becomes a Logistic Regression model.



Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$





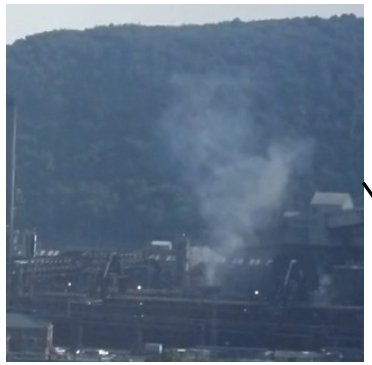
Binary cross-entropy loss

$$L(y, \hat{y}) = -\sum_{c=1}^{2} y_c \log \hat{y}_c \text{ for 2 classes}$$

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{2} y_c^{(i)} \log \hat{y}_c{}^{(i)}$$

Cross-entropy loss -- https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

Logistic Regression model fits a logistic curve to the data to perform classification tasks.



Has smoke (class 1)

No smoke (class 0)

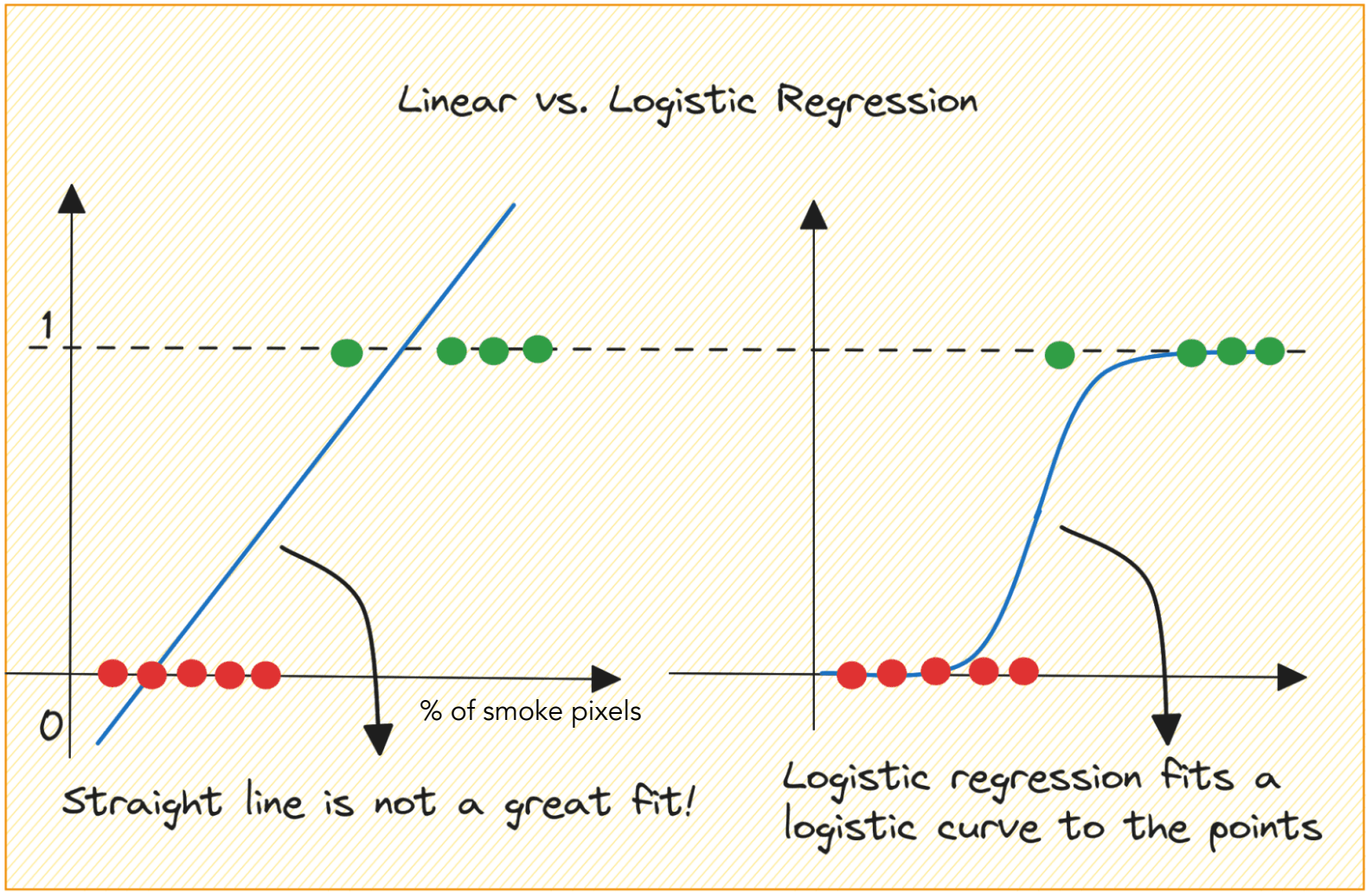


Figure source -- https://www.kdnuggets.com/building-predictive-models-logistic-regression-in-python

Many activation functions exist for various purposes. For example, in classification, ReLU is typical for the middle layers, and sigmoid (or softmax) is for the final layer.

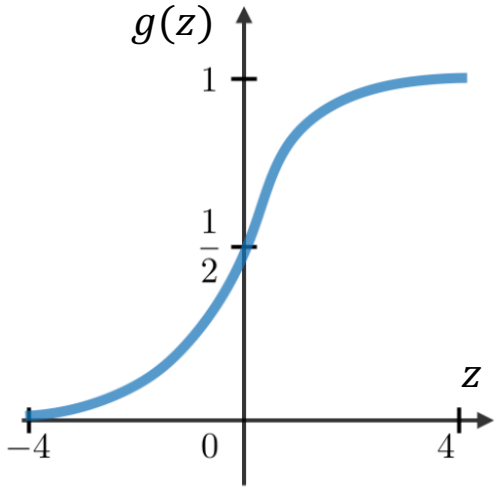
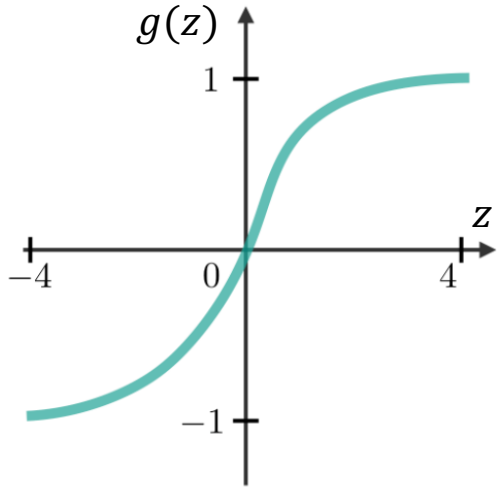
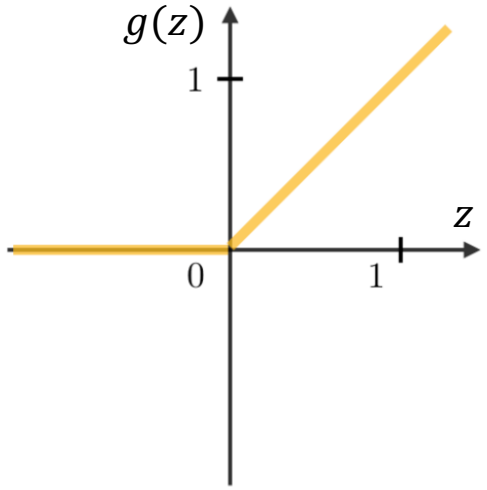
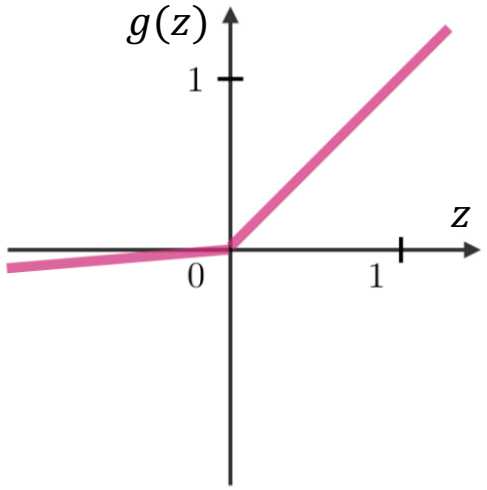
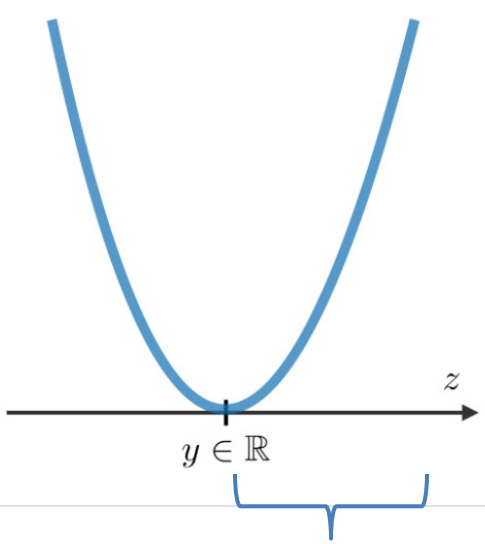
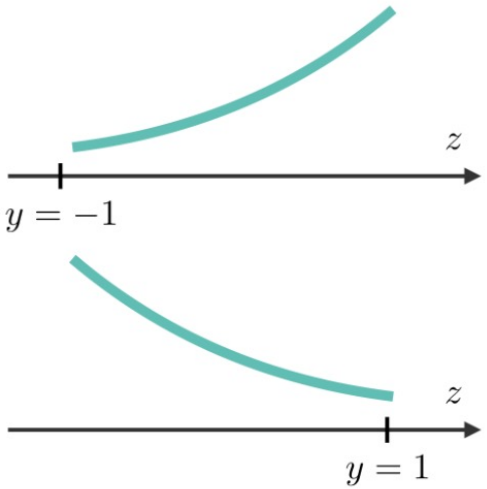
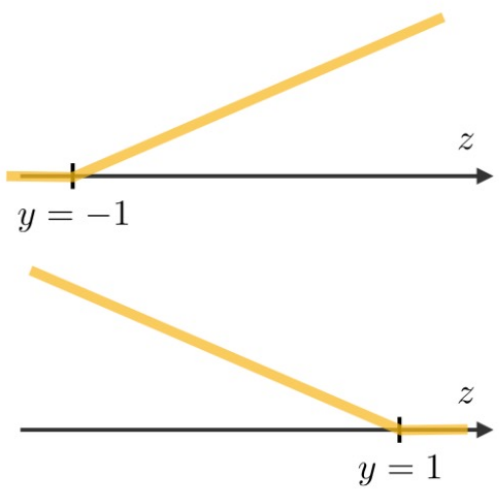
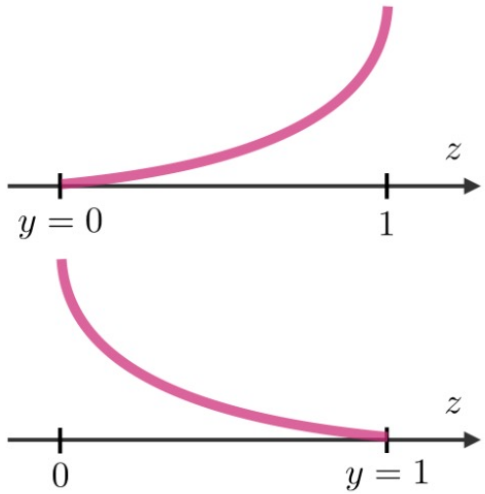
Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

Figure source -- https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-deep-learning#nn

Also, there are different loss functions for different types of tasks. For example, the least squared error is for regression, and the others below are for classification.

Least squared error	Logistic loss	Hinge loss	Cross-entropy
$\frac{1}{2}(y-z)^2$	$\log(1+\exp(-yz))$	$\max(0, 1-yz)$	$-\left[y\log(z)+(1-y)\log(1-z)\right]$
			

How different is z from y, where y is the ground truth, and z is the prediction.

Figure source -- https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-supervised-learning#notations

The major reason of using activation functions is to introduce non-linearity.

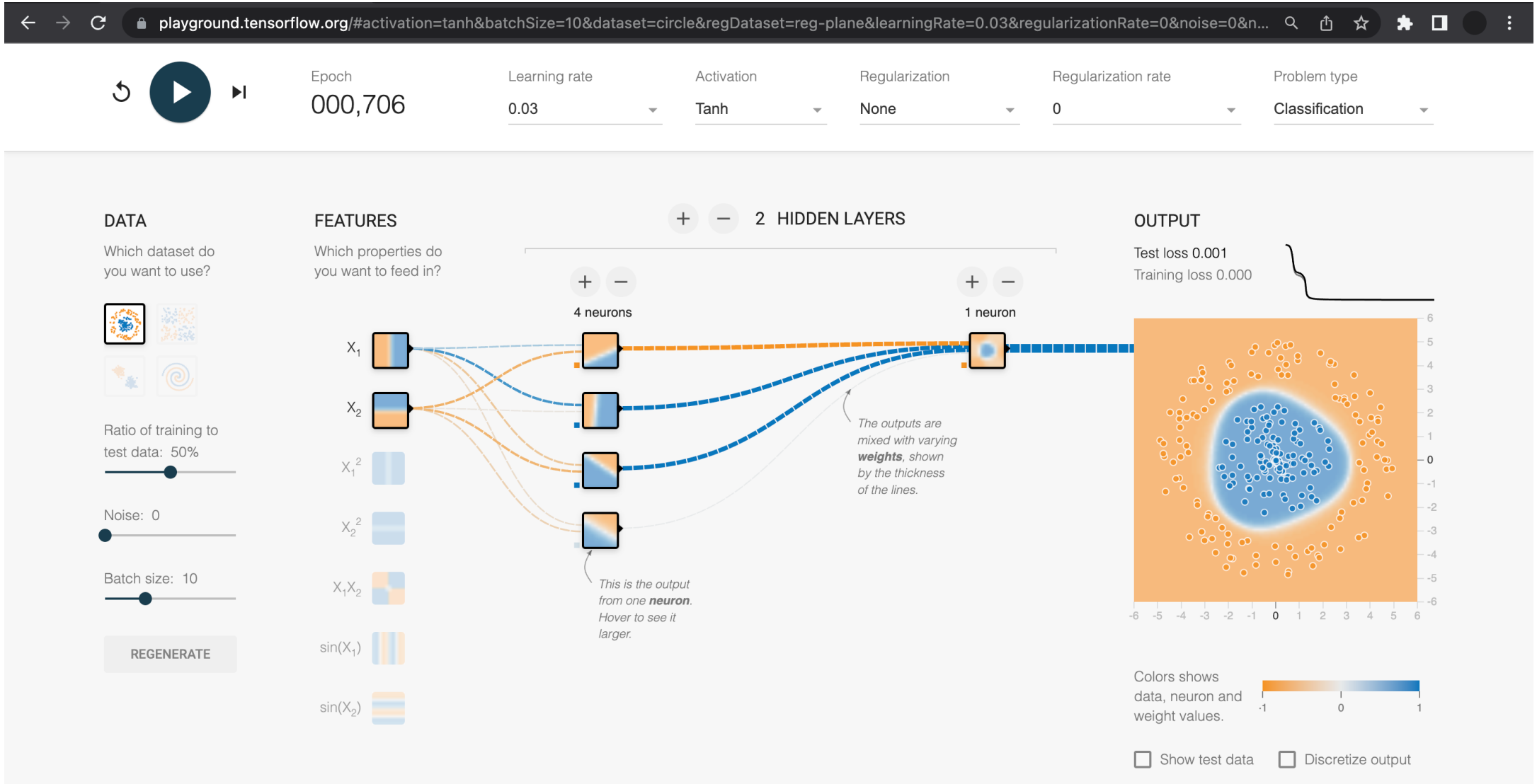


Figure source -- the neural network playground using the Tanh activation function

For example, the identify activation function does not help in grouping non-linear data.

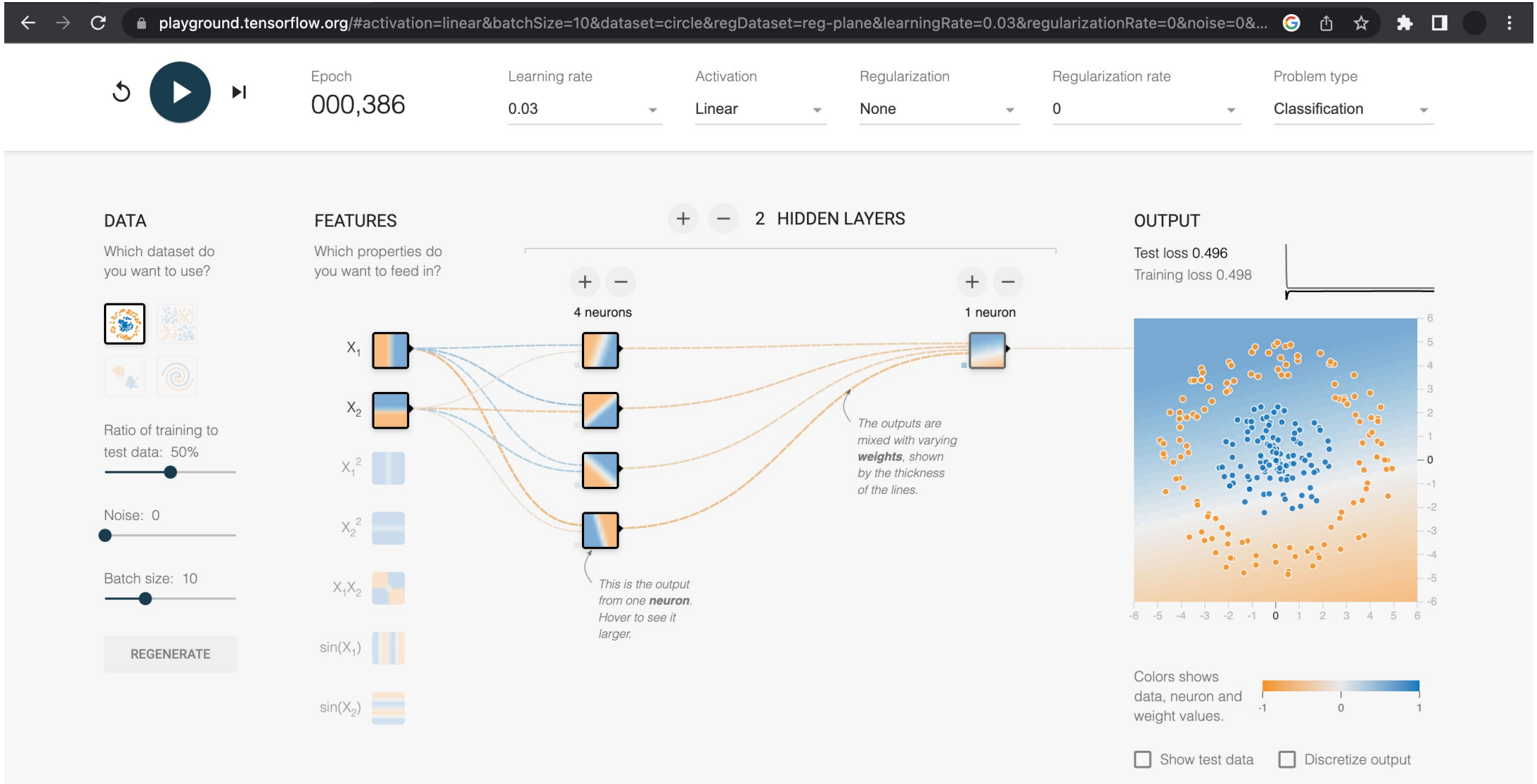
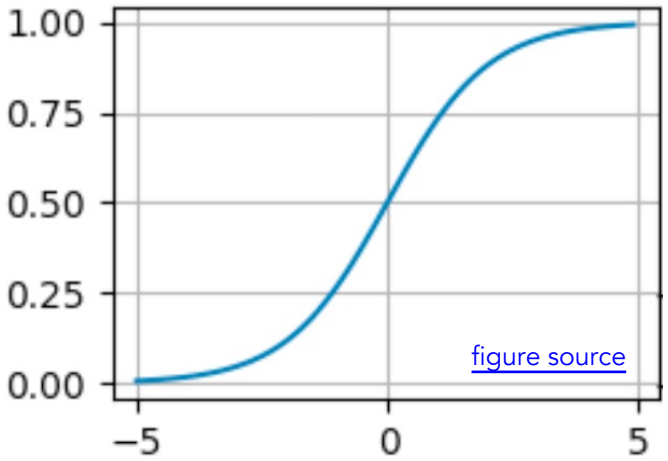


Figure source -- [the neural network playground using the identity activation function]

Why do we use a sigmoid function (or similar) as the activation function σ for the classification task?

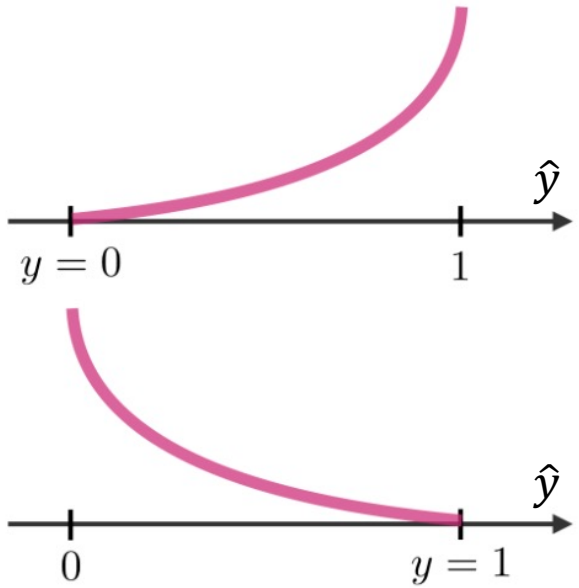
Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$



Hint: think about the following:

- $\hat{y} = \sigma(w^T x + b)$
- What is the role of σ?
- What does the predicted output $\hat{y}$ look like?

Why do we use cross entropy as the loss function for the classification task?



Hint: think about the difference between entropy and cross entropy:

$$\text{Entropy} = \sum_{\text{Class } c} y_c \cdot \log_2\left(\frac{1}{y_c}\right)$$

$$\text{Cross Entropy} = \sum_{\text{Class } c} y_c \cdot \log_2\left(\frac{1}{\hat{y}_c}\right)$$

We can think about cross entropy loss as the exam, the true value y_c as the teacher (from the training data), and the predicted value $\hat{y}_c$ as the student (from the model).

$$\text{Cross Entropy} = \sum_{\text{Class } c} y_c \cdot \log_2\left(\frac{1}{\hat{y}_c}\right)$$

Exam

Teacher

Student

How can we train the deep neural network?

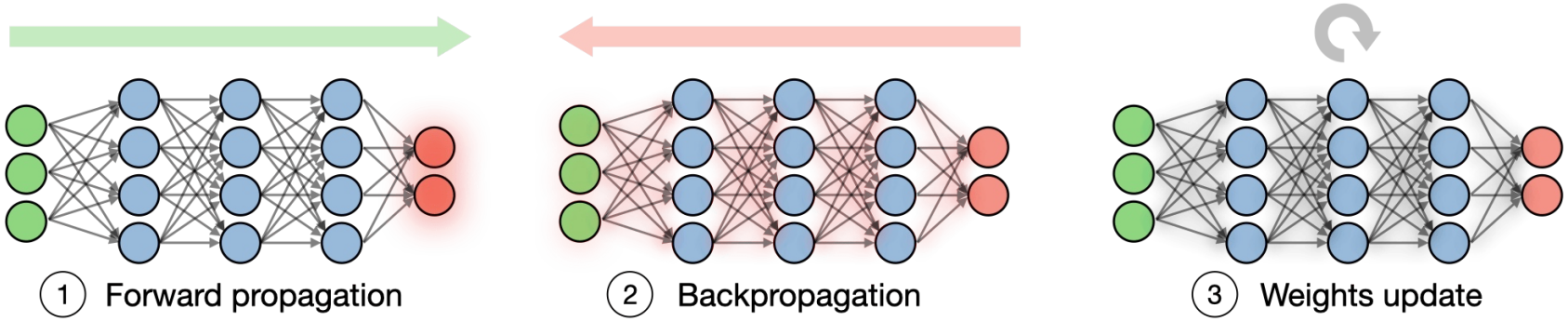
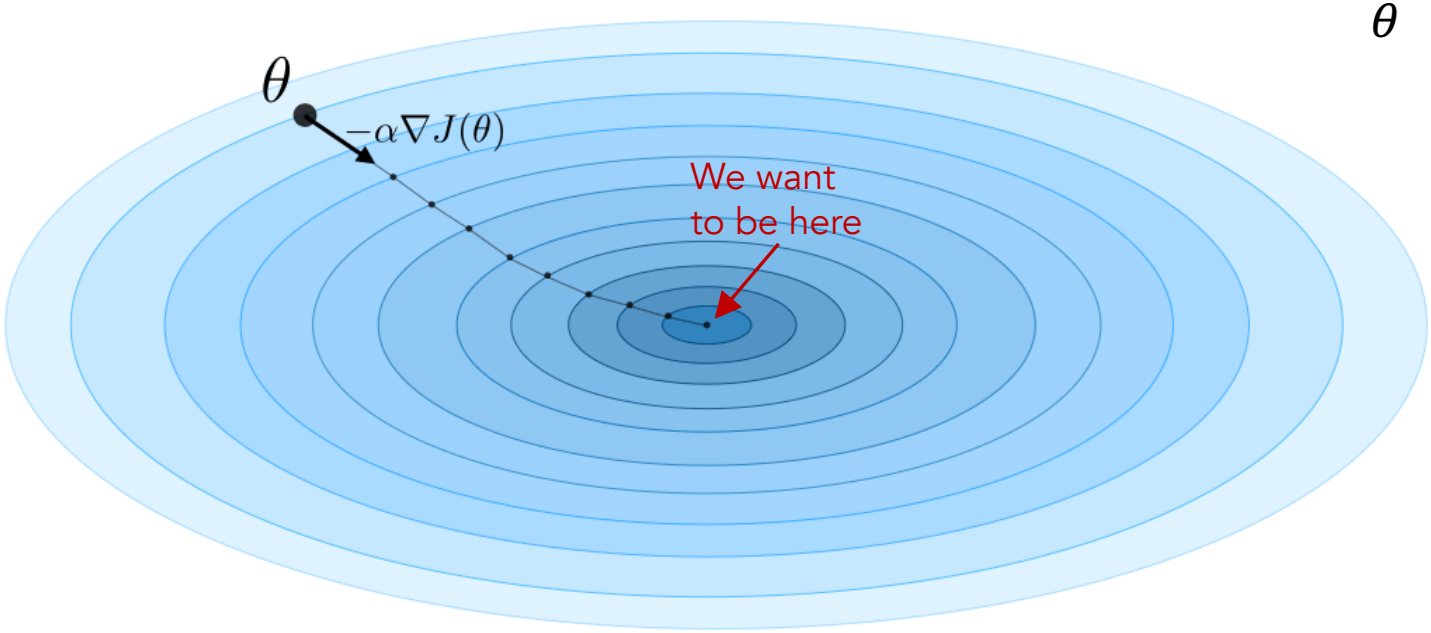


Figure source -- https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-deep-learning-tips-and-tricks#running-nn

We need to use optimization algorithms, such as gradient descent, to help us find a local minimum (or global minimum for convex functions) on the cost function. We need to set a learning rate, which means the pace of moving forward for each update.



$$\min_{\theta} J(\theta) = \min_{\theta} \frac{1}{n} \sum_{i=1}^{n} L\left(y^{(i)}, \hat{y}^{(i)}\right)$$

$$\theta_{t+1} \leftarrow \theta_t - \alpha \cdot \nabla J(\theta_t)$$

θ: model parameters

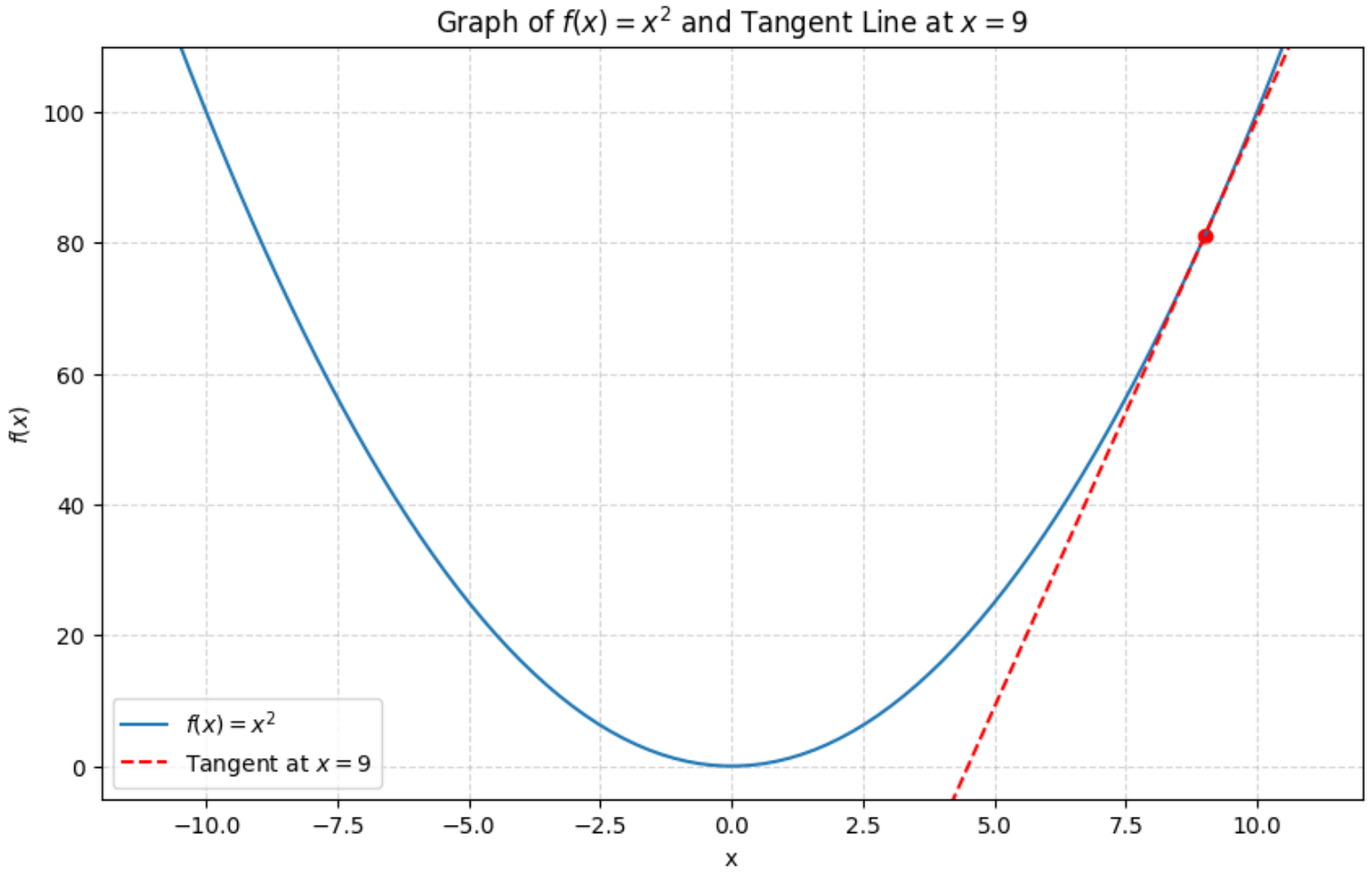
α: learning rate

t: training step

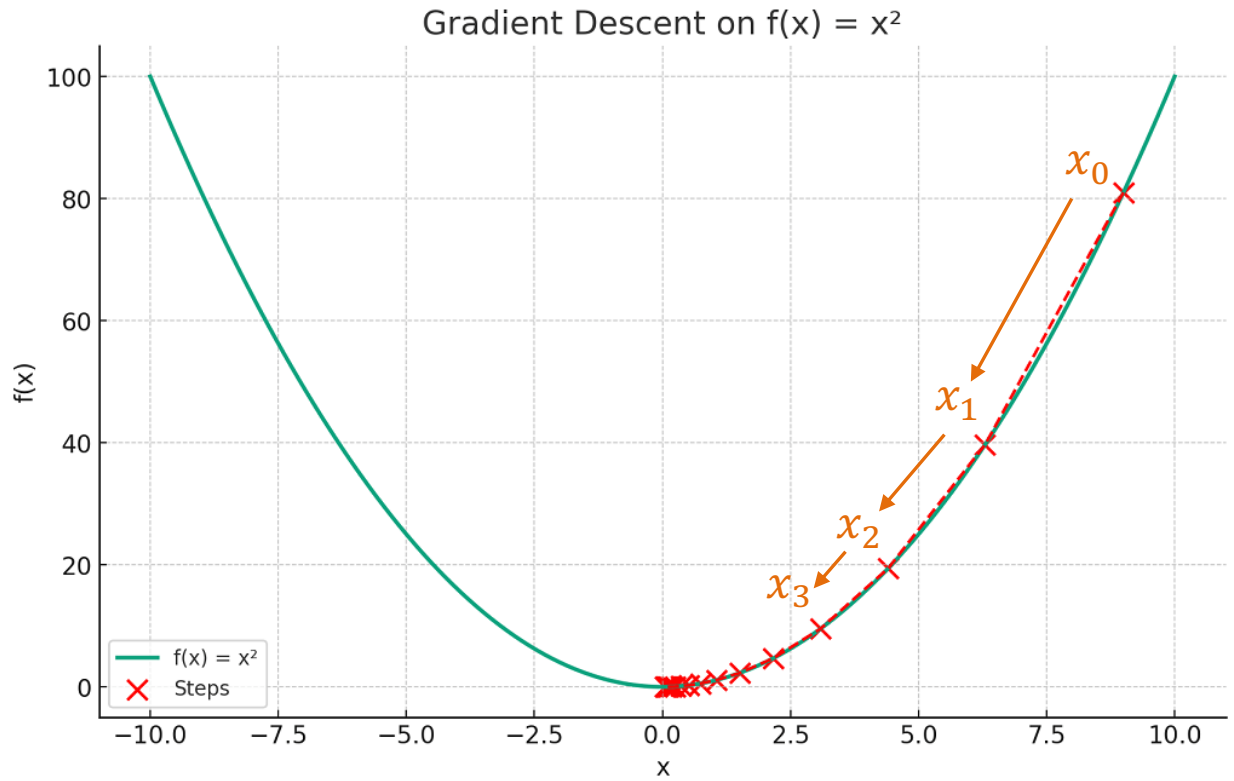
$\nabla J(\theta)$: gradient

Figure source -- https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-supervised-learning#notations

Gradient is a generalization of derivative. The intuition is that computing the derivative of $f(x)$ at point x_t means computing the slope of the tangent line to $f(x)$ at point x_t.



Exercise 5.1: Perform three gradient descent updates (i.e., compute x_1, x_2, and x_3) for one parameter x with the cost function $f(x) = x^2$, using the starting point $x_0 = 9$ and learning rate $\alpha = 0.1$ (with the provided gradient function $\nabla f(x)$ below).



$$f(x) = x^2$$

$$\nabla f(x) = 2x$$

$$x_{t+1} \longleftarrow x_t - \alpha \cdot \nabla f(x_t)$$

x: model parameter

α: learning rate

t: training step

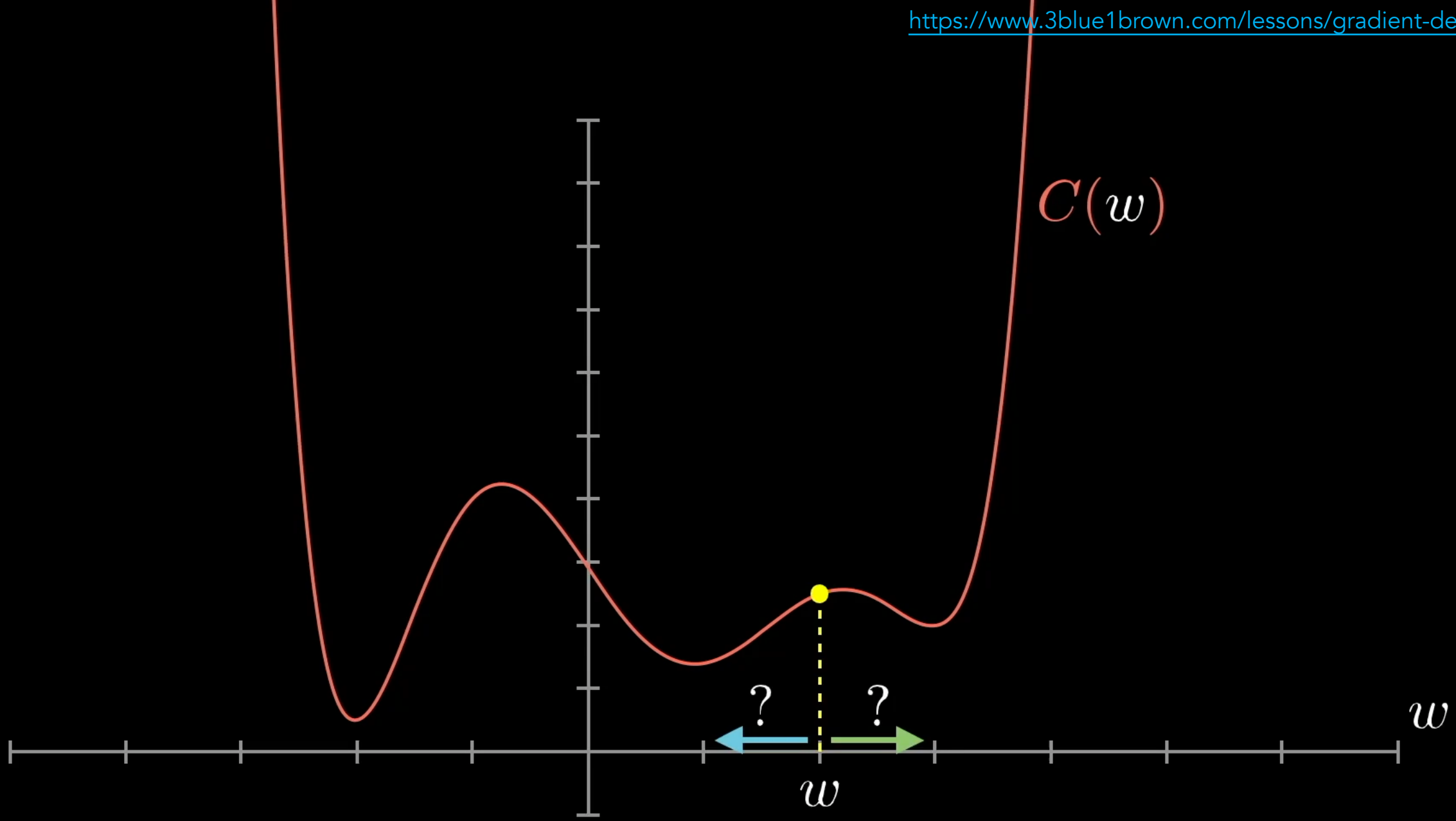
$\nabla f(x)$: gradient

Figure source -- https://vitalflux.com/gradient-descent-explained-simply-with-examples/

Why do we need gradient descent? Why not just taking the derivative of $J(\theta)$ and setting it to zero to find the minimum (like what we did for linear regression)?

$$\min_{\theta} J(\theta) = \min_{\theta} \frac{1}{n} \sum_{i=1}^{n} L\left(y^{(i)}, \hat{y}^{(i)}\right)$$

https://www.3blue1brown.com/lessons/gradient-descent



We can reach a local minimum of the loss function by following the gradient (i.e., slope).

We need to adjust the learning rate strategically. A large learning rate could lead to divergent behavior in the model. For example, the training loss could wiggle.

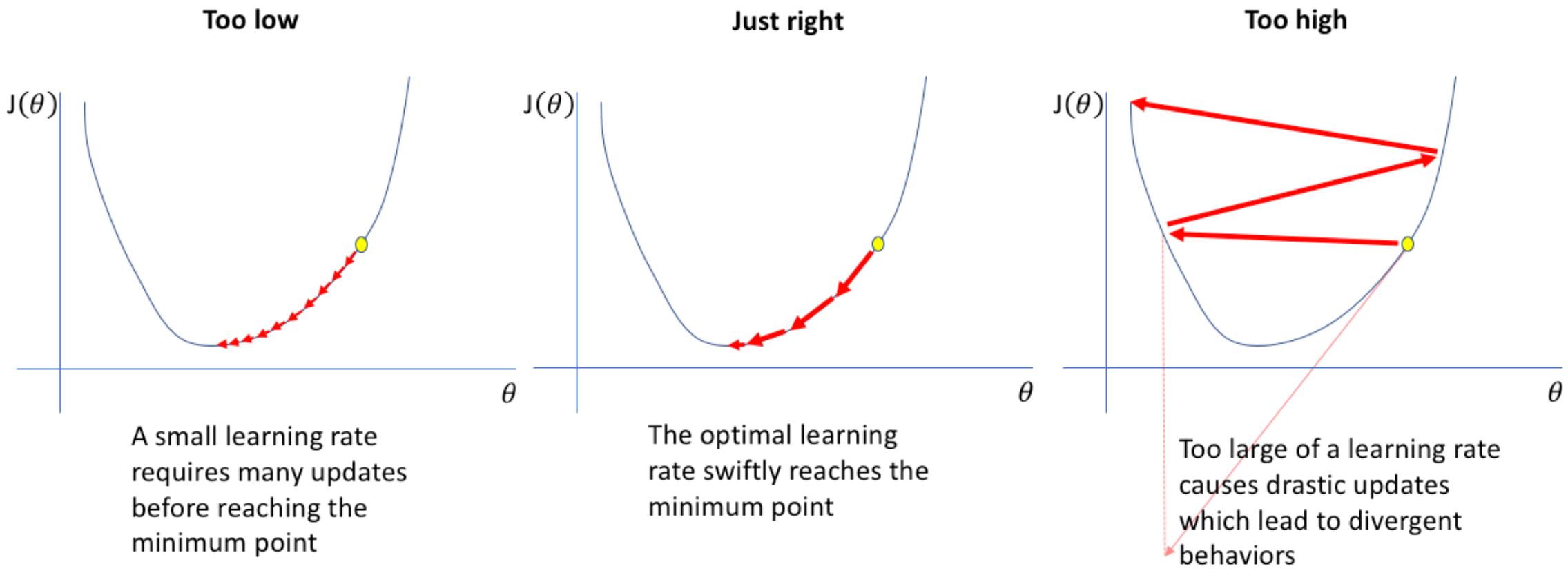
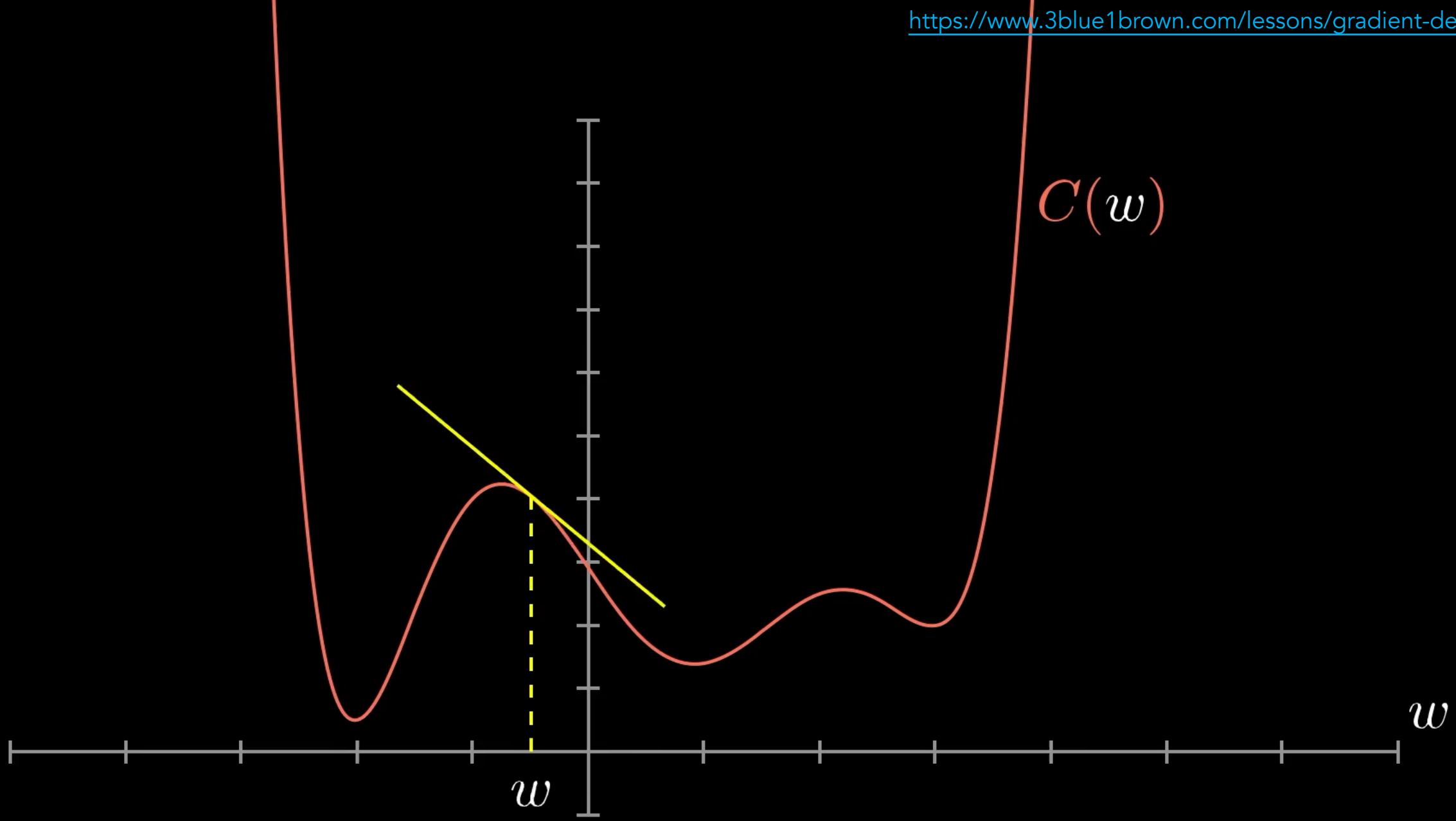


Figure source -- https://www.jeremyjordan.me/nn-learning-rate/

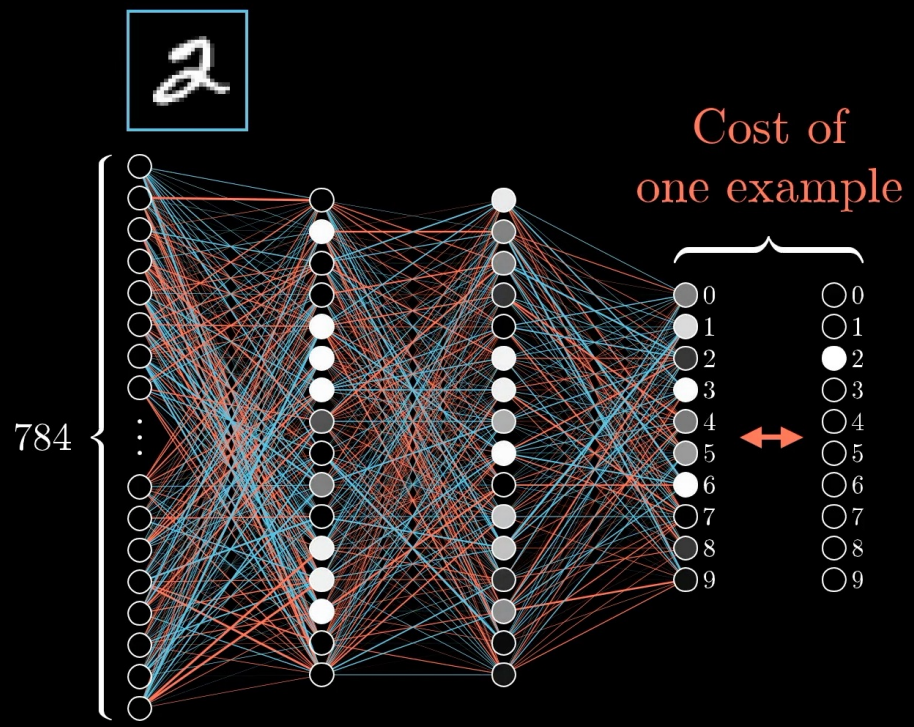
https://www.3blue1brown.com/lessons/gradient-descent



Take smaller steps (i.e., smaller learning rate) when the slope is getting small.

How do we adjust the weight after computing the loss for each iteration? To update the weights in previous layers, we need to use the backpropagation algorithm.

Compute the loss (error) for each iteration



Backpropagate the errors

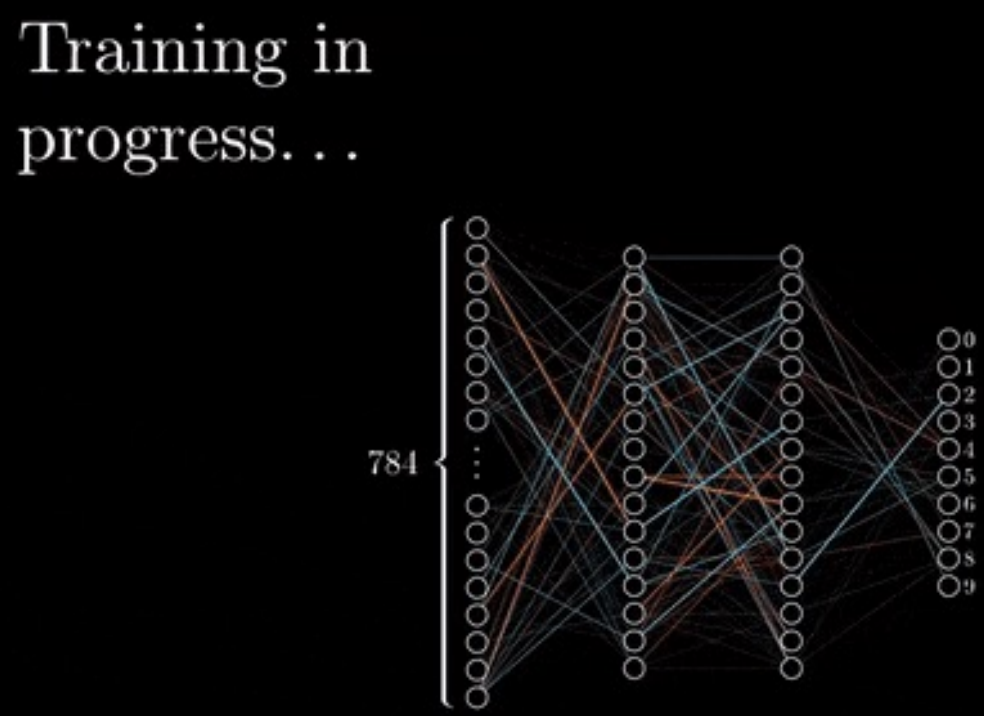


Figure source -- https://www.3blue1brown.com/lessons/backpropagation

Intuitively speaking, after comparing the prediction and the ground truth, we want to increase the weight for the neuron we care about the most and decrease the others.

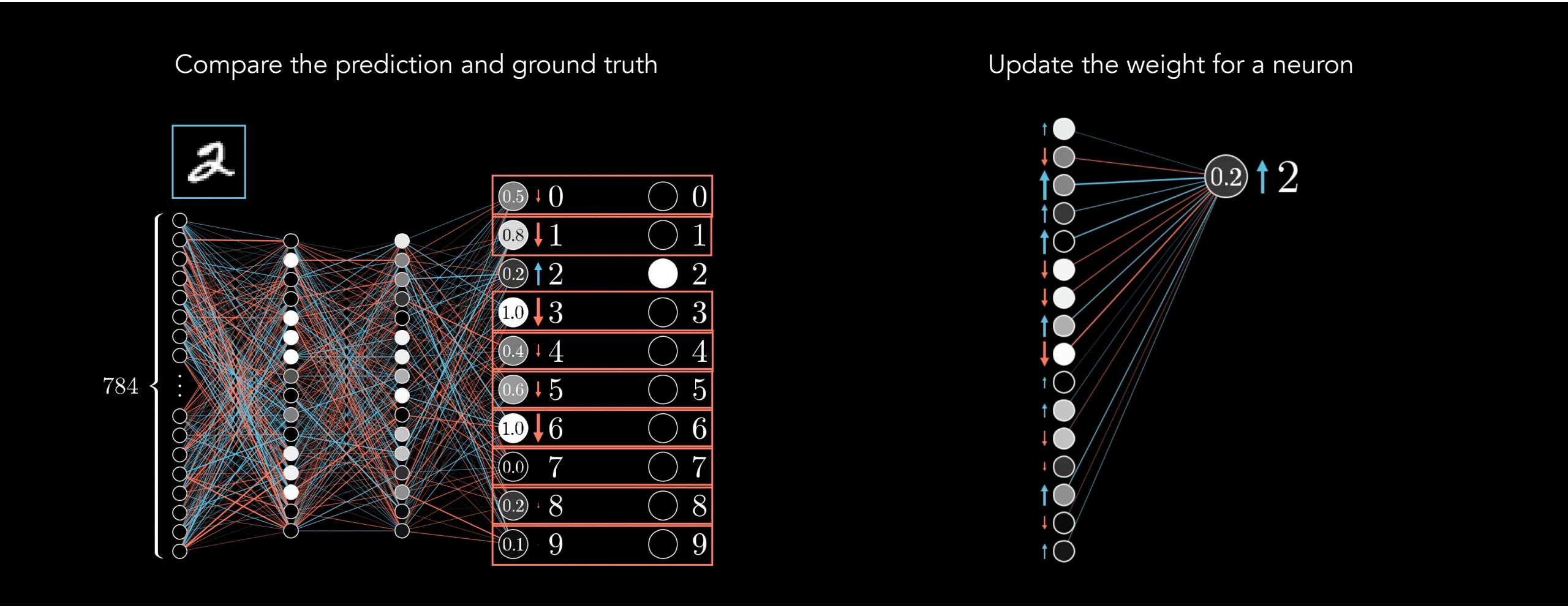
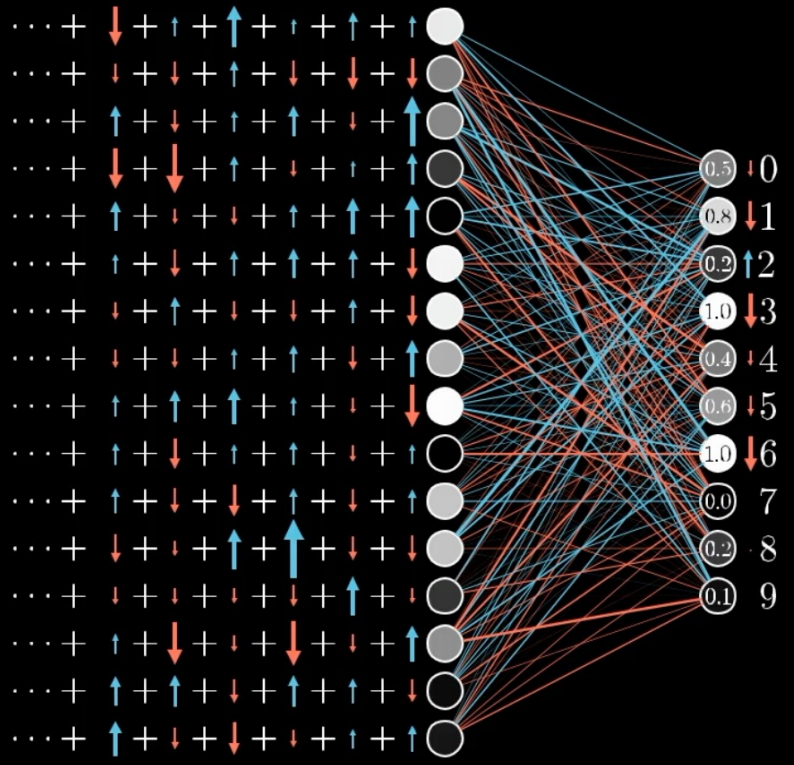


Figure source -- https://www.3blue1brown.com/lessons/backpropagation

We apply the same idea to iteratively update all the weights for all the neurons in every previous layer, starting from the last layer, and backpropagate the errors back.

Update the weights for the current layer



Update the weights for the previous layer

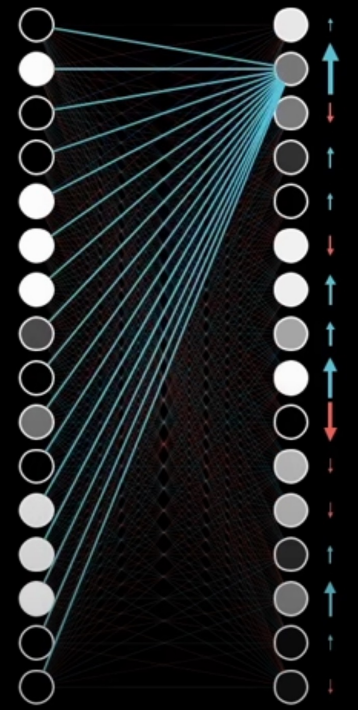


Figure source -- https://www.3blue1brown.com/lessons/backpropagation

In practice, we use mini-batches (instead of all data) when running gradient descent to increase the speed (and save computer memory) when updating neuron weights.

Divide data randomly into mini-batches

Use mini-batches to approximate the original gradient

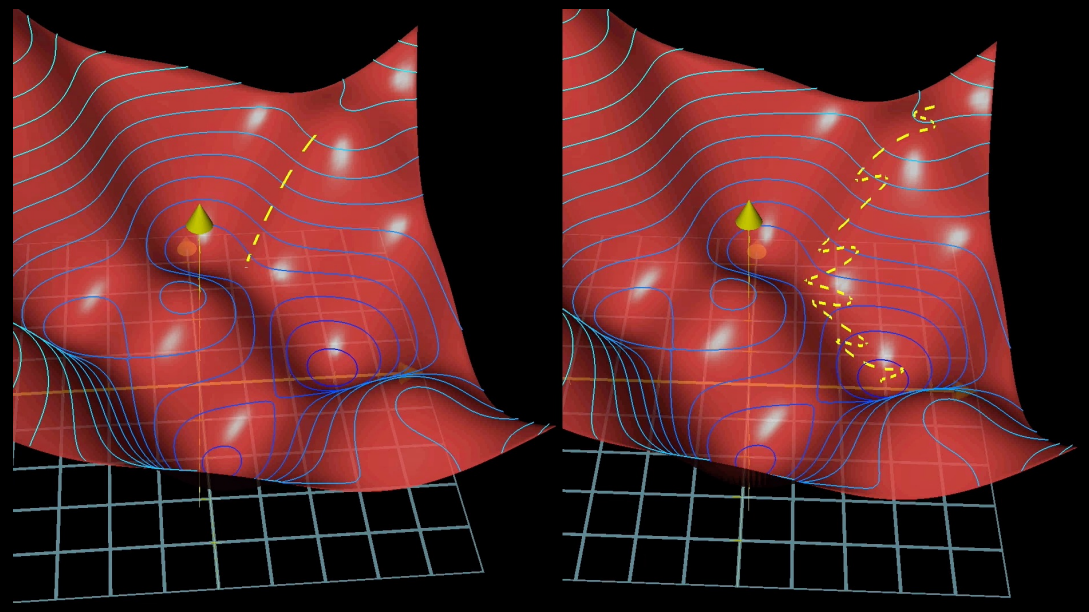


Figure source -- https://www.3blue1brown.com/lessons/backpropagation

The backpropagation algorithm applies the chain rule in calculus to compute gradient.

$$\frac{\partial g(f(\theta))}{\partial \theta} = \frac{\partial g}{\partial f} \cdot \frac{\partial f}{\partial \theta}$$

$$\nabla J(\theta_1) = \frac{\partial J(\theta)}{\partial \theta_1} = \frac{\partial J(\theta)}{\partial a^{(3)}} \cdot \frac{\partial a^{(3)}}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial \theta_1}$$

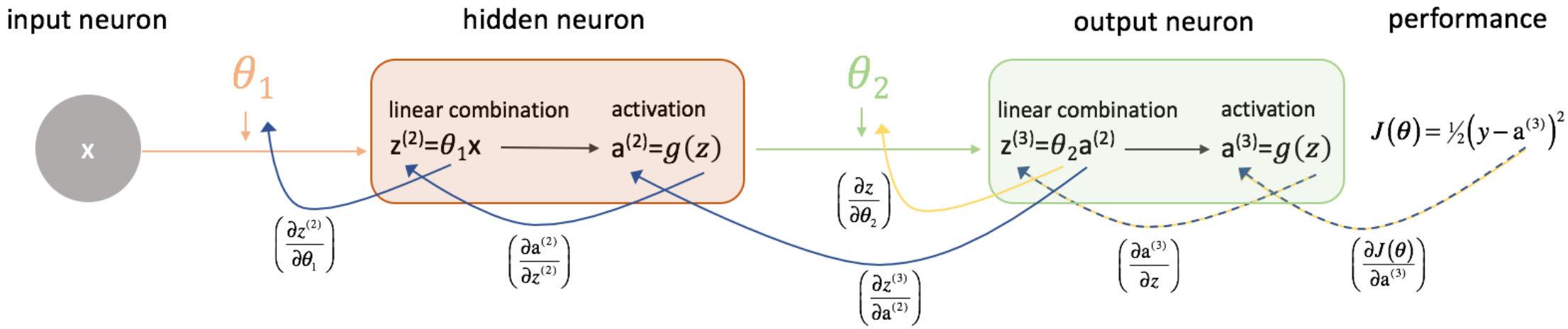


Figure source and more about backpropagation -- https://www.jeremyjordan.me/neural-networks-training/

Deep neural nets can overfit easily due to a huge number of parameters. How can we deal with overfitting?

We can combat overfitting by randomly dropping out neurons with a pre-defined probability (i.e., the dropout technique), which forces the model to avoid paying too much attention to a particular set of features.

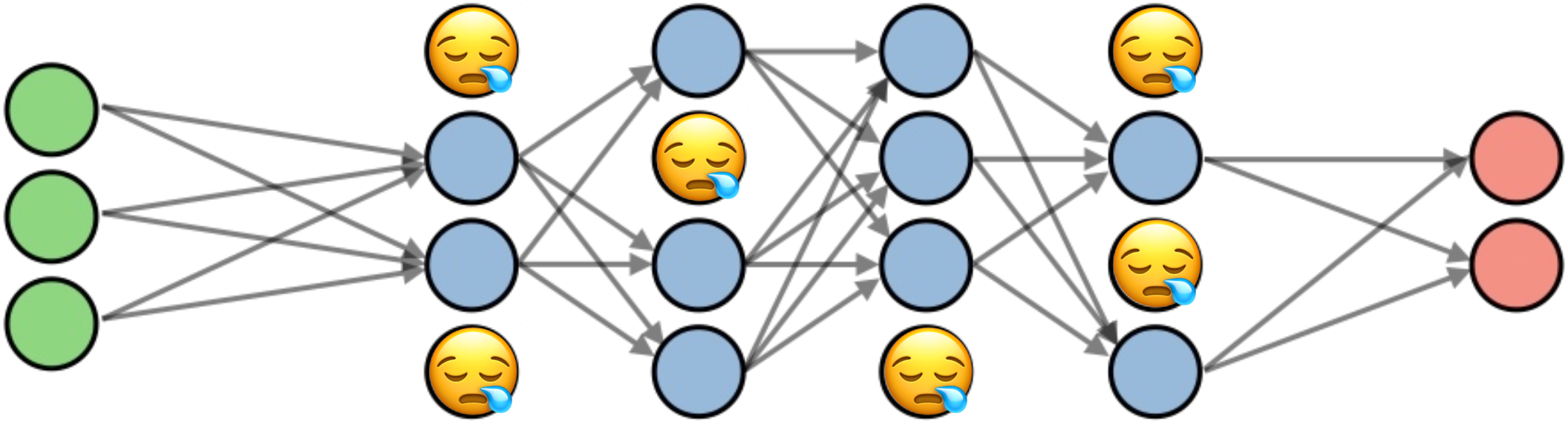


Figure source -- https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-deep-learning-tips-and-tricks#regularization

We can also combat overfitting by using the regularization technique (also setting its strength factor λ), which regulates model weights to ensure that they are not too large.

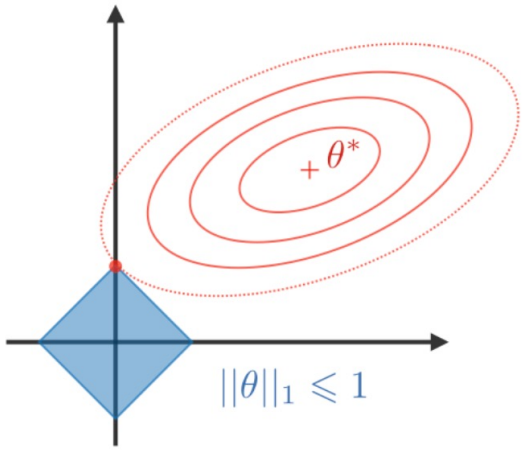
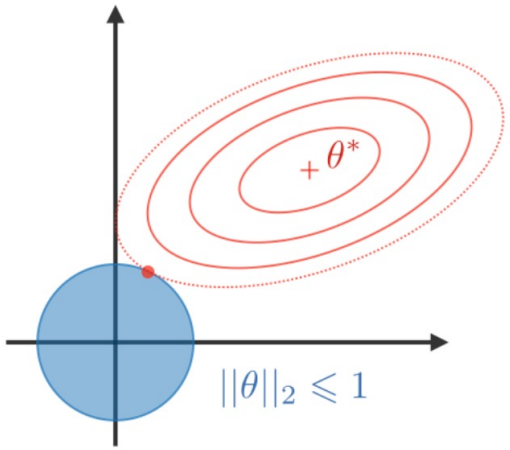
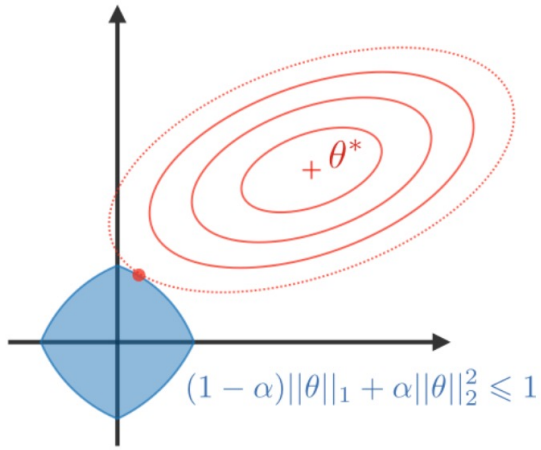
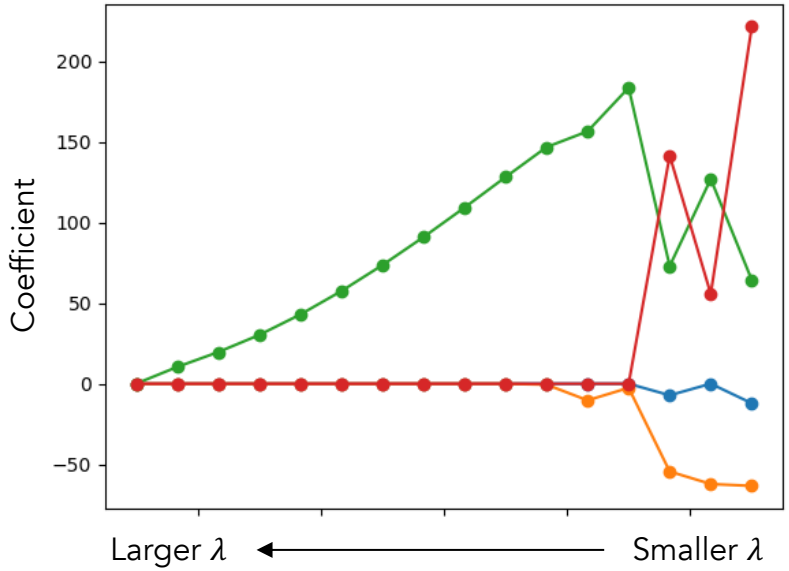
LASSO	Ridge	Elastic Net
• Shrinks coefficients to 0 • Good for variable selection	Makes coefficients smaller	Tradeoff between variable selection and small coefficients
		
$\ldots + \lambda\|\|\theta\|\|_1$ $\lambda \in \mathbb{R}$	$\ldots + \lambda\|\|\theta\|\|_2^2$ $\lambda \in \mathbb{R}$	$\ldots + \lambda\left[(1-\alpha)\|\|\theta\|\|_1 + \alpha\|\|\theta\|\|_2^2\right]$ $\lambda \in \mathbb{R}, \alpha \in [0,1]$

Figure source -- https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-deep-learning-tips-and-tricks#regularization

Increasing regularization strength λ shrinks the coefficients. Lasso specifically shrinks the coefficients to zero (which can be used for feature selection), while Ridge only shrink the coefficients to very small numbers that are close to zero (but not exactly zero).

Lasso Regularization (L1 norm)

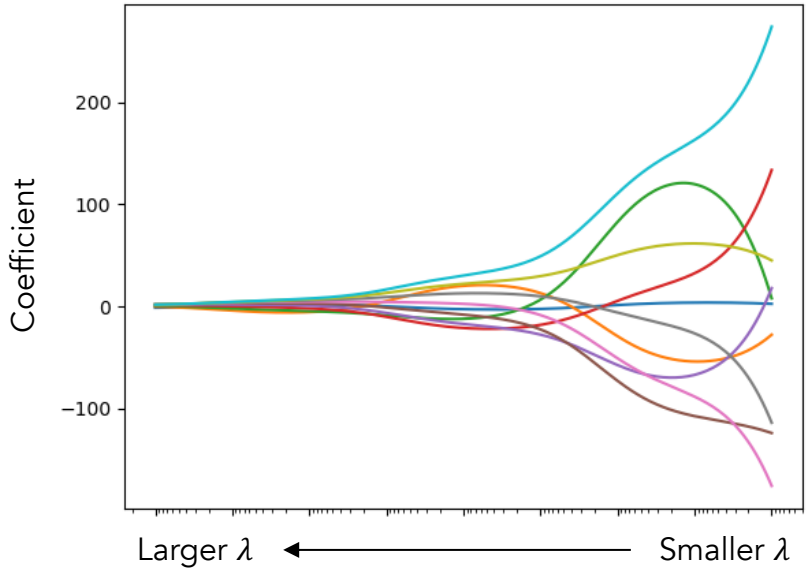


$$\min_{\theta} J(\theta) + \lambda \|\theta\|_1$$

$$\|\theta\|_1 = |\theta_1| + |\theta_2| + \cdots + |\theta_p|$$

Source: https://scikit-learn.org/stable/auto_examples/linear_model/plot_logistic_path.html

Ridge Regularization (L2 norm)

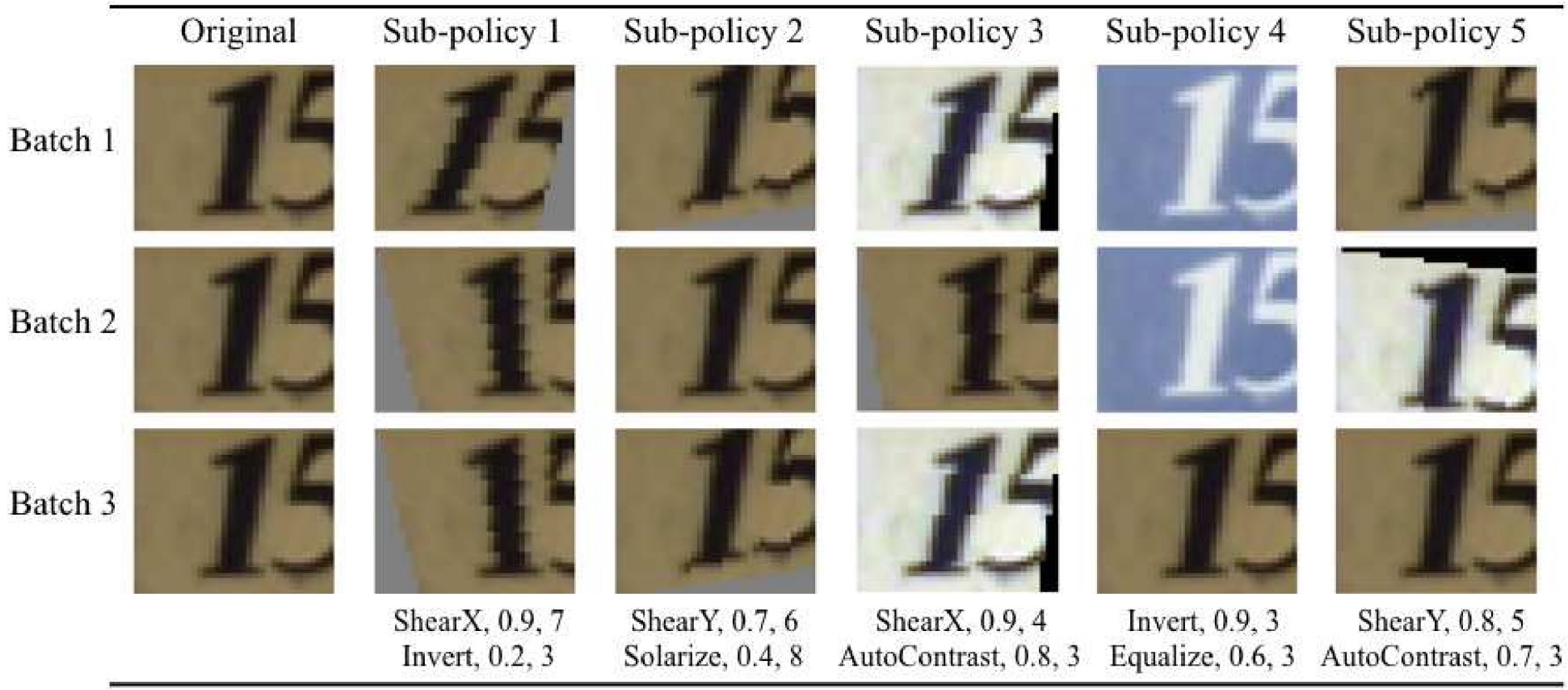


$$\min_{\theta} J(\theta) + \lambda \|\theta\|_2$$

$$\|\theta\|_2 = \sqrt[2]{(\theta_1)^2 + (\theta_1)^2 + \cdots + (\theta_p)^2}$$

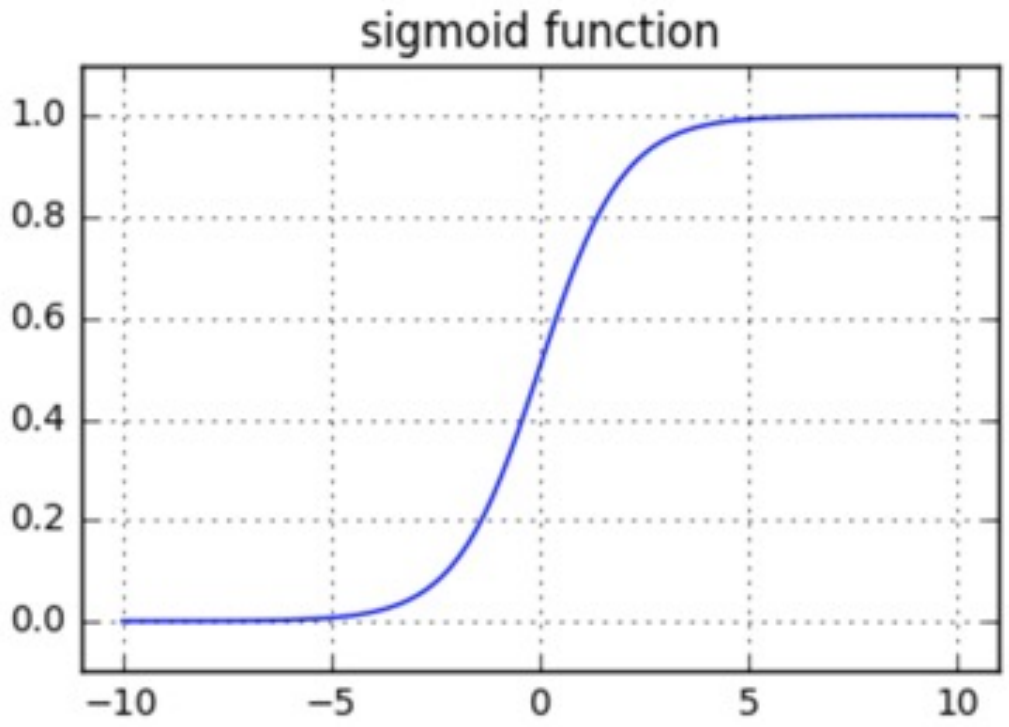
Source: https://scikit-learn.org/stable/auto_examples/linear_model/plot_ridge_path.html

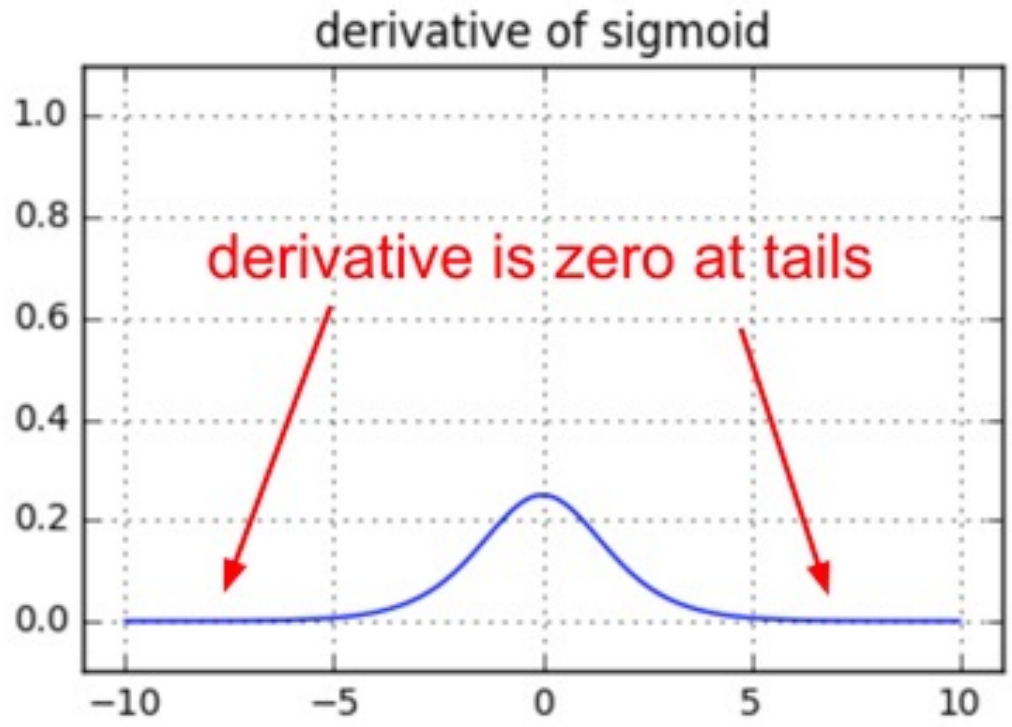
We can also augment our data on the fly to combat overfitting (i.e., the so-called data augmentation technique), such as randomly rotating, cropping, and changing colors.



Cubuk, E. D., et al. (2019). AutoAugment: Learning augmentation strategies from data. In Proceedings of CVPR conference.

Deep learning models can suffer from vanishing gradient, where the gradient becomes too small during backpropagation, and thus the model weights are hard to update. The example below shows the problem when using the sigmoid activation function.

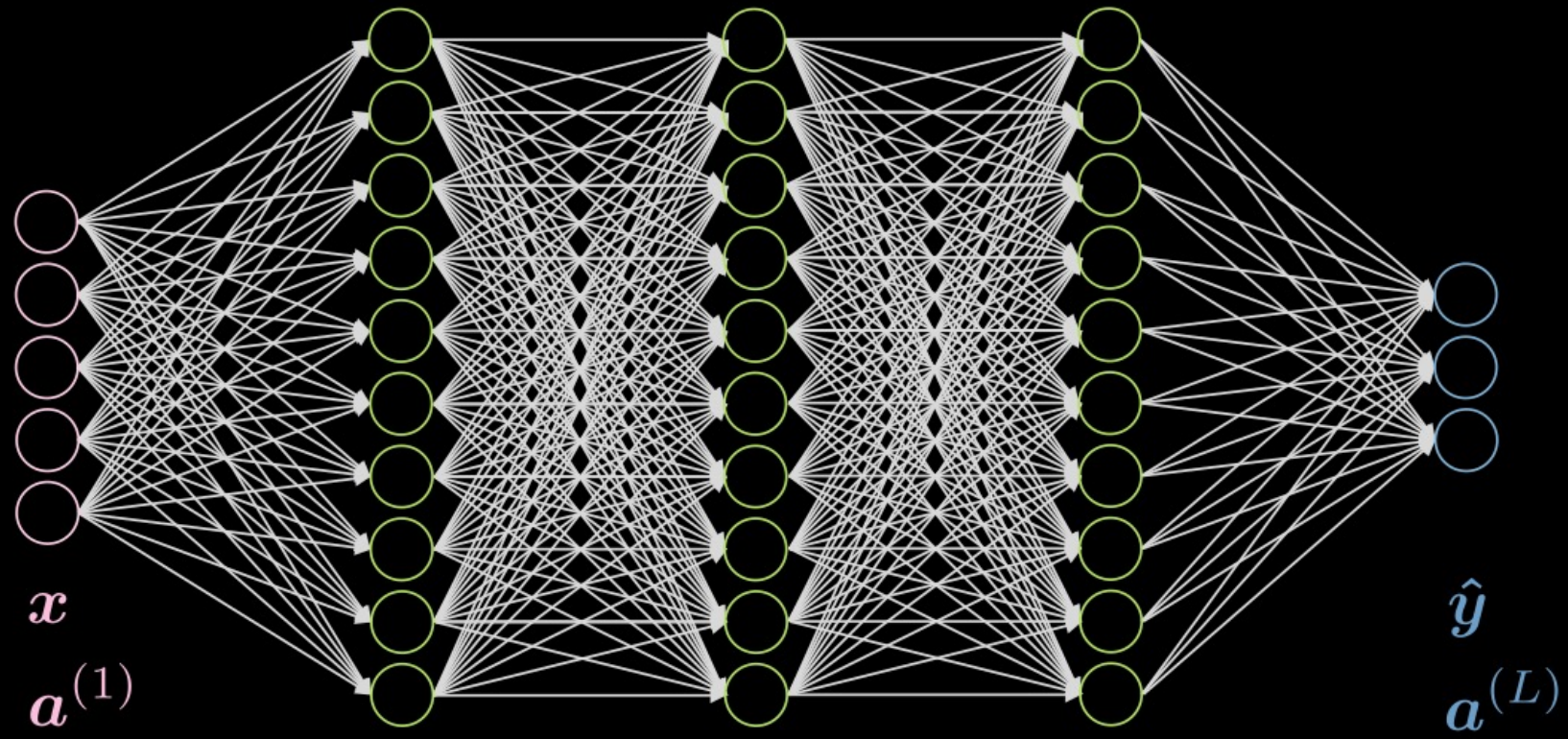




More about problems in the gradient -- https://karpathy.medium.com/yes-you-should-understand-backprop-e2f06eab496b

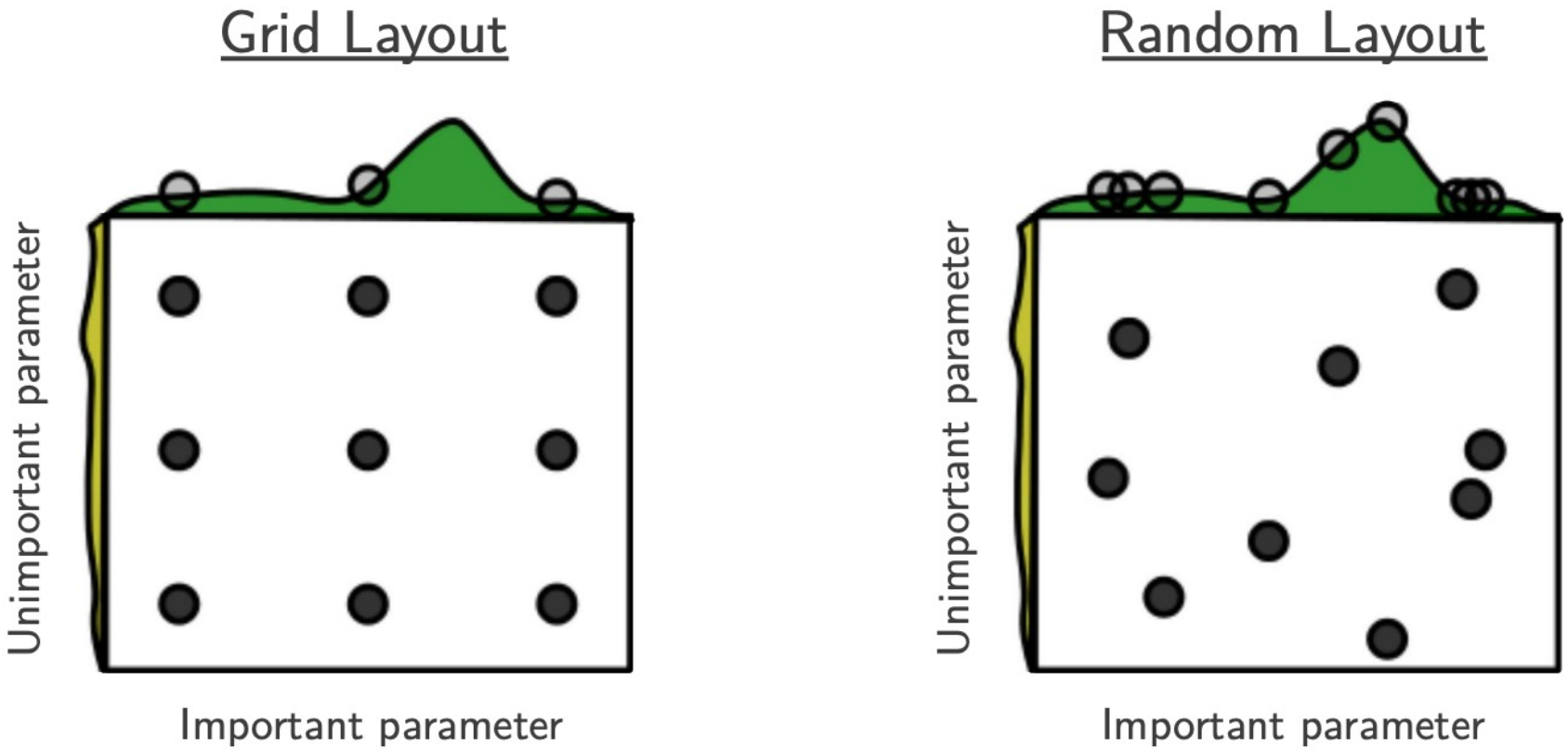
During model training, we need to tune hyperparameters, which means the parameters that are set prior to the learning process (unlike model parameters, i.e., model weights).

- Learning rate?
- Batch size?
- Layer size?
- Number of layers?
- Dropout rate?
- Strength of regularization?
- Activation function?
- Number of epoch?
- …



More info -- https://medium.com/pytorch/accelerate-your-hyperparameter-optimization-with-pytorchs-ecosystem-tools-bc17001b9a49

Common hyperparameter tunning strategies involve grid or random search. Random search is typically considered more effective when there are a lot of hyperparameters.



Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of machine learning research, 13(2).

Take-Away Messages

- Deep learning is the idea of stacking many layers of artificial neurons.
- You can change the activation function and loss function to perform different tasks.
- The activation function can transform the weighted sum of the input non-linearly.
- The loss function measures the distance between the ground truth and the prediction.
- Gradient descent is used to help us find a local minimum on the error (cost) function.
- When performing gradient descent, we need to set a learning rate to determine the pace.
- We need to use the backpropagation algorithm to iteratively update the weights in previous layers.
- Using dropout, regularization, or data augmentation can help us combat overfitting.

Questions?