Introduction to Neural Networks

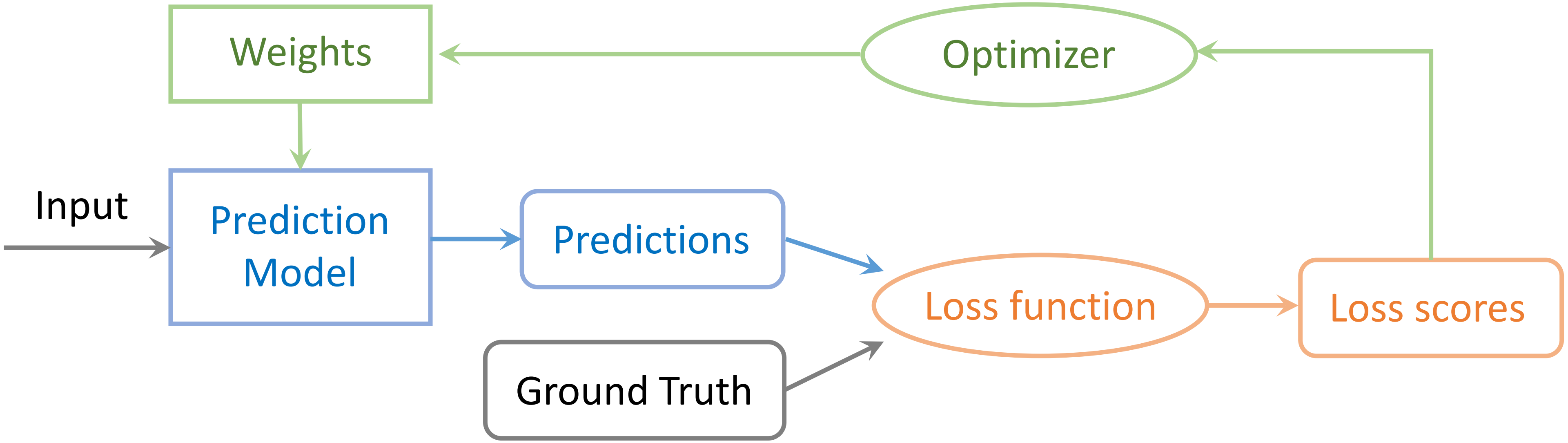
Chia-Po Wei

Department of Electrical Engineering
National Sun Yat-sen University

Outline

- Optimization
  - Gradient Descent
  - Backpropagation
- Neural Networks
  - Input/Hidden/Output Layers
  - Keras for Training Neural Networks

Training Pipeline



- The training pipeline consists of choosing the prediction model, the loss function, and the optimizer.
- Once these choices are made, we can feed the input data and labels to start the training process.

Gradient Descent

```
keras.optimizers.SGD(lr=0.001)
```

- Consider the following sequence

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \alpha \nabla f(\mathbf{w}_n), n \geq 0$$

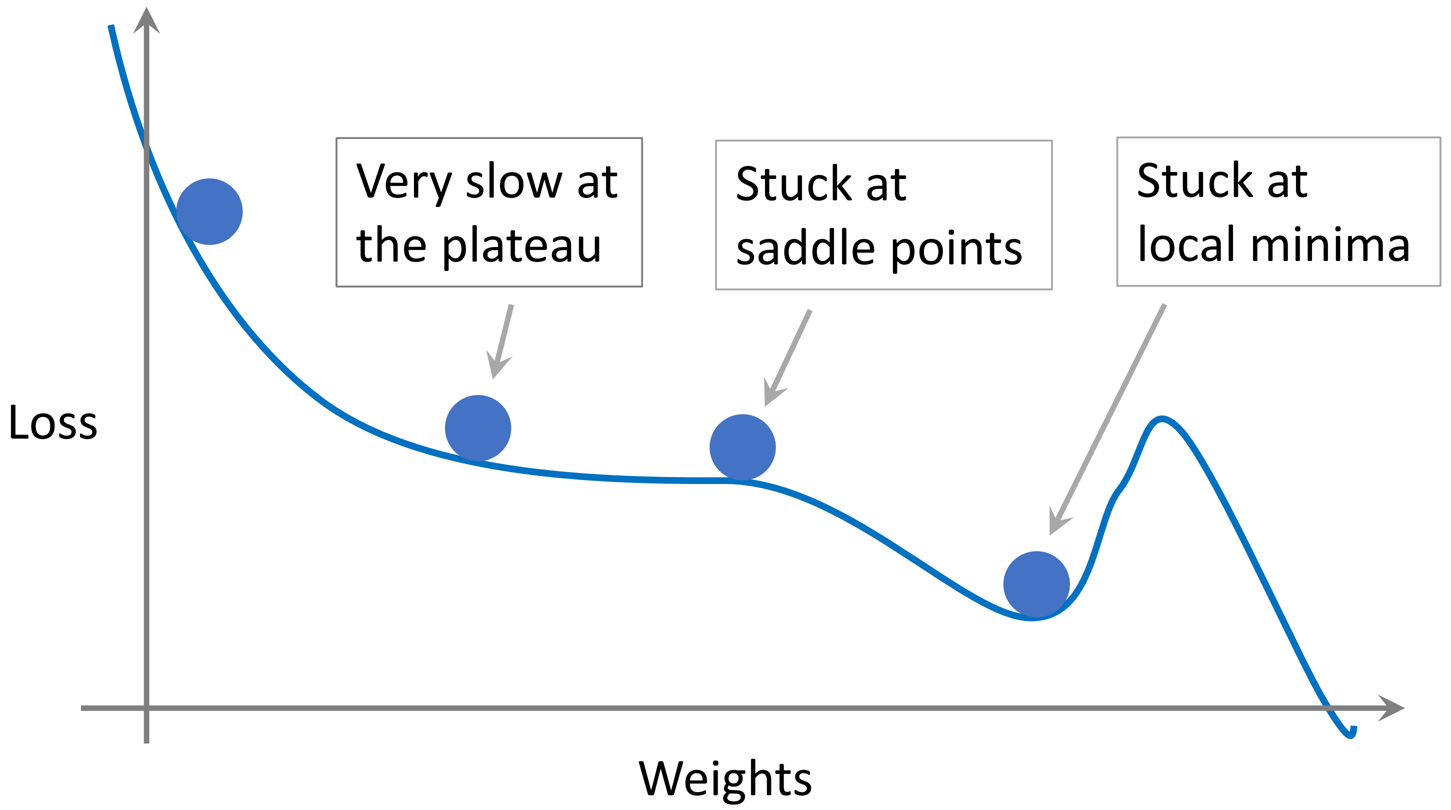
- Since $-\nabla f(\mathbf{w}_n)$ is the negative gradient at $\mathbf{w}_n$, we have

$$f(\mathbf{w}_0) \geq f(\mathbf{w}_1) \geq f(\mathbf{w}_2) \geq \cdots$$

- The scalar α is called the **learning rate**.
- How to properly select the learning rate?
- Try the Adam optimizer

```
keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999)
```

Gradient Descent (cont.)

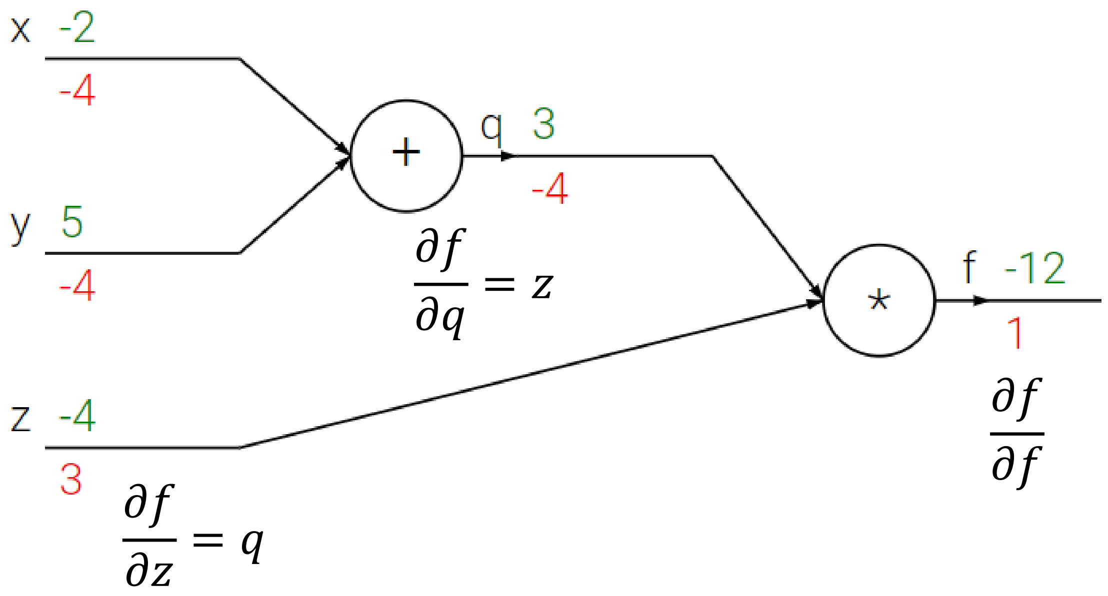


[slide credit: Hung-yi Lee]

Backpropagation

- $f(x, y, z) = (x + y)z$
- Let $x = -2, y = 5, z = -4$
- Define $q = x + y$, $\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$
- Then $f = qz$, $\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$

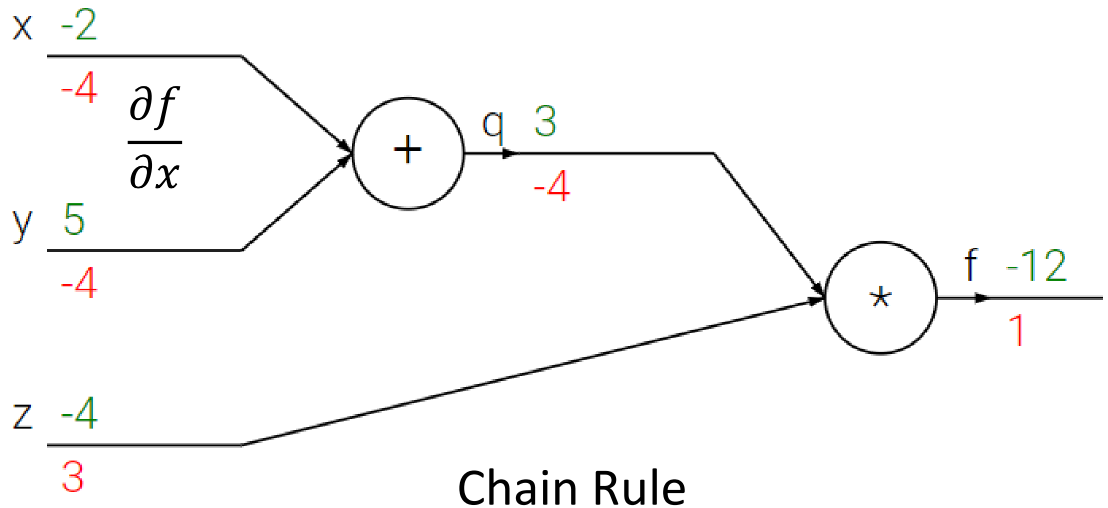
Calculate $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation (cont.)

- $f(x, y, z) = (x + y)z$
- Let $x = -2, y = 5, z = -4$
- Define $q = x + y, \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$
- Then $f = qz, \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$

Calculate $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



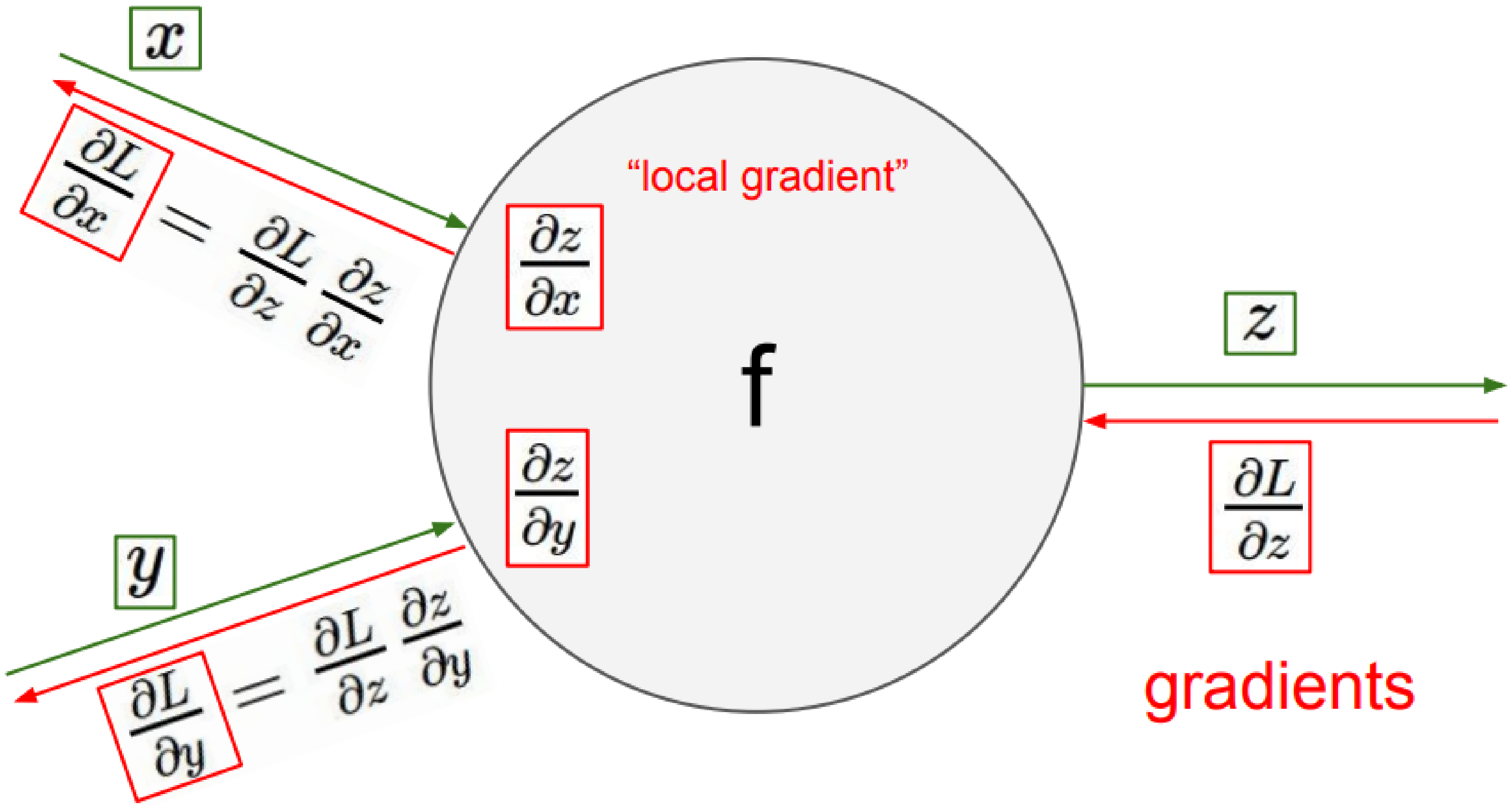
Chain Rule

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

Upstream gradient

Local gradient

Backpropagation (cont.)



[slide credit: Stanford CS231n]

Neural Networks

- Linear Model: $f(\mathbf{x}) = \mathbf{W}\mathbf{x}$
- 2-layer Neural Network:

$$f(\mathbf{x}) = f_2(\mathrm{ReLU}(f_1(\mathbf{x}))) = \mathbf{W}_2\max(0, \mathbf{W}_1\mathbf{x})$$

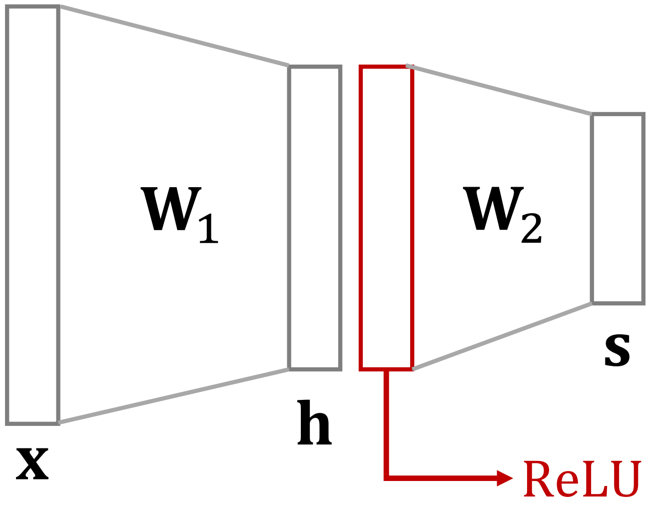
$$f_1(\mathbf{x}) = \mathbf{W}_1\mathbf{x}, \quad f_2(\mathbf{h}) = \mathbf{W}_2\mathbf{h}, \ \mathrm{ReLU}(\mathbf{h}) = \max(0, \mathbf{h})$$

Neural Networks (cont.)

- 2-layer Neural Network:

$$f(\mathbf{x}) = f_2(\mathrm{ReLU}(f_1(\mathbf{x}))) = \mathbf{W}_2 \max(0, \mathbf{W}_1\mathbf{x})$$

$$f_1(\mathbf{x}) = \mathbf{W}_1\mathbf{x}, \quad f_2(\mathbf{h}) = \mathbf{W}_2\mathbf{h}, \; \mathrm{ReLU}(\mathbf{h}) = \max(0, \mathbf{h})$$



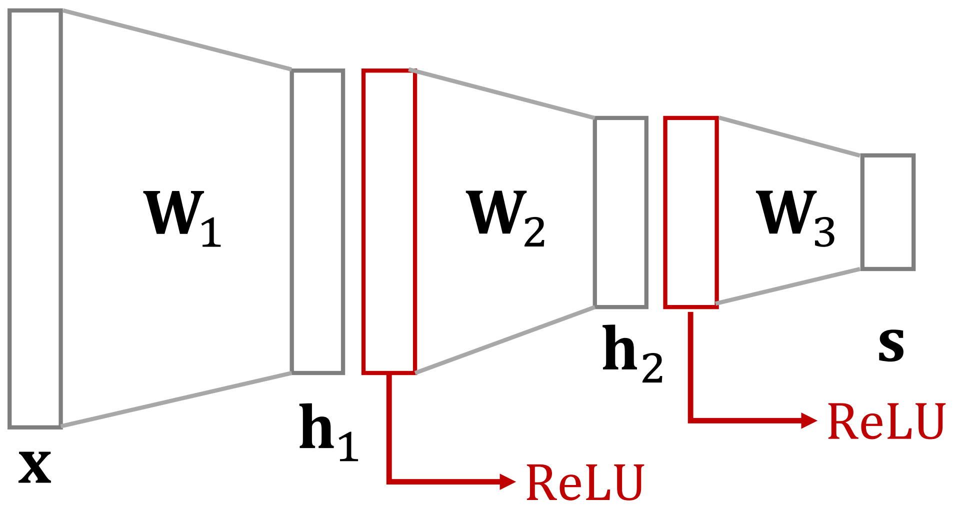
- Take CIFAR10 for example
- $\dim(\mathbf{x})$ is 32x32x3 = 3072
- $\dim(\mathbf{s})$ is 10 (number of classes)
- $\dim(\mathbf{h})$ is manually defined

Neural Network (cont.)

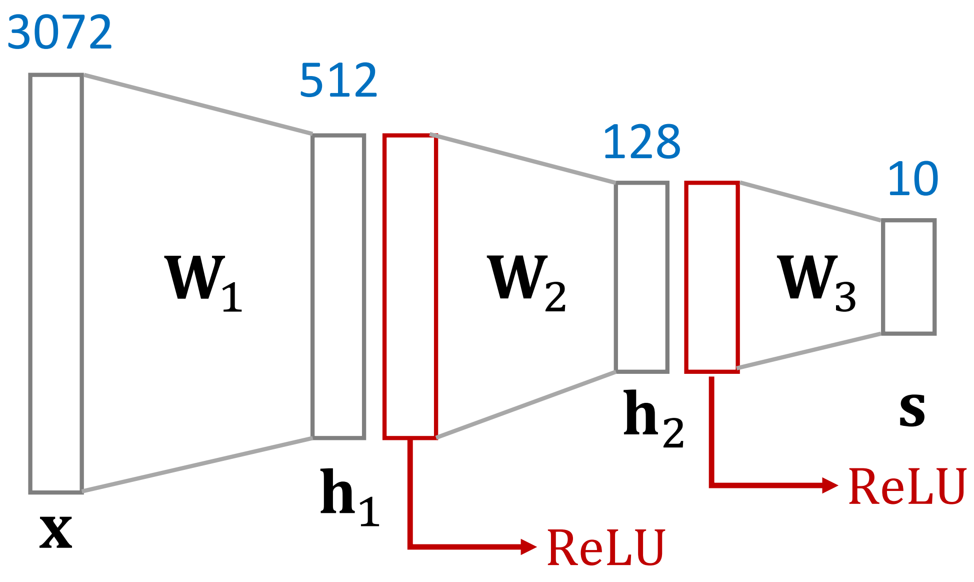
- 3-layer Neural Network:

$$f(\mathbf{x}) = f_3(\mathrm{ReLU}(f_2(\mathrm{ReLU}(f_1(\mathbf{x}))) = \mathbf{W}_3 \max(0, \mathbf{W}_2 \max(0, \mathbf{W}_1 \mathbf{x}))$$

$$f_1(\mathbf{x}) = \mathbf{W}_1 \mathbf{x}, \;\; f_2(\mathbf{h}_1) = \mathbf{W}_2 \mathbf{h}_1, \;\; f_3(\mathbf{h}_2) = \mathbf{W}_3 \mathbf{h}_2, \mathrm{ReLU}(\mathbf{h}) = \max(0, \mathbf{h})$$



Keras: Creating Models via Sequential()



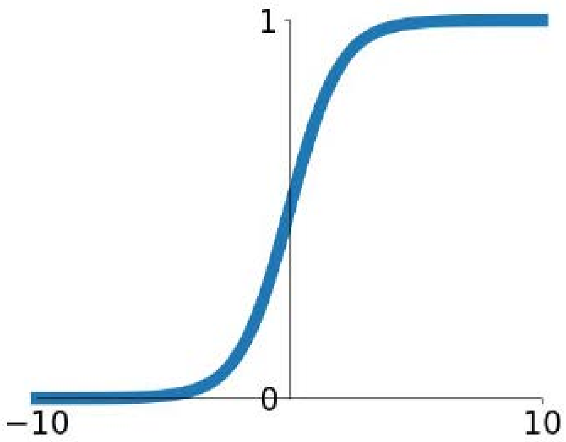

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(32, 32, 3)),
    keras.layers.Dense(512, activation=tf.nn.relu),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)])
```

Hyperparameters:

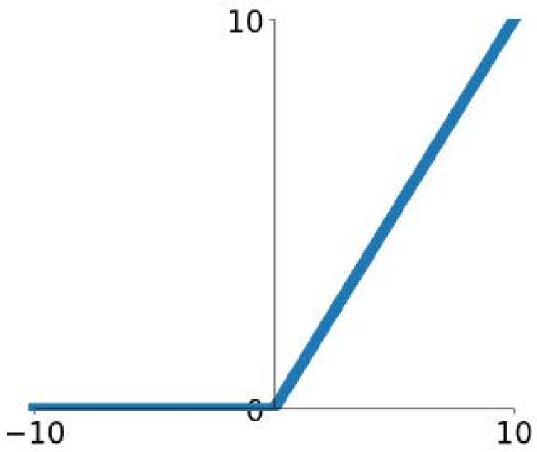
- The number of layers
- The number of neurons per
- The activation functions

Activation Functions

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



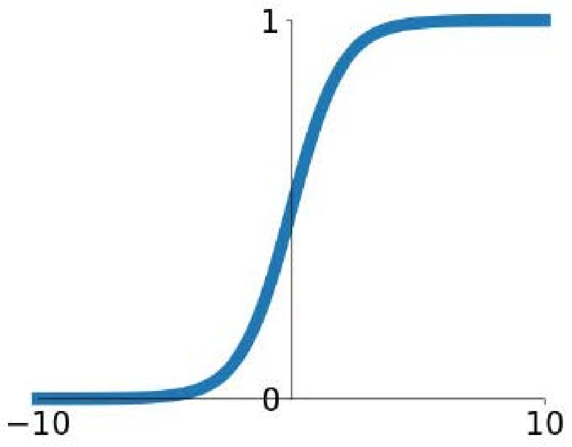
ReLU: $\max(0, x)$



- Sigmoid leads to the vanishing gradient problem (seldom used).
- ReLU is now widely used in the architecture design of neural networks.

Sigmoid

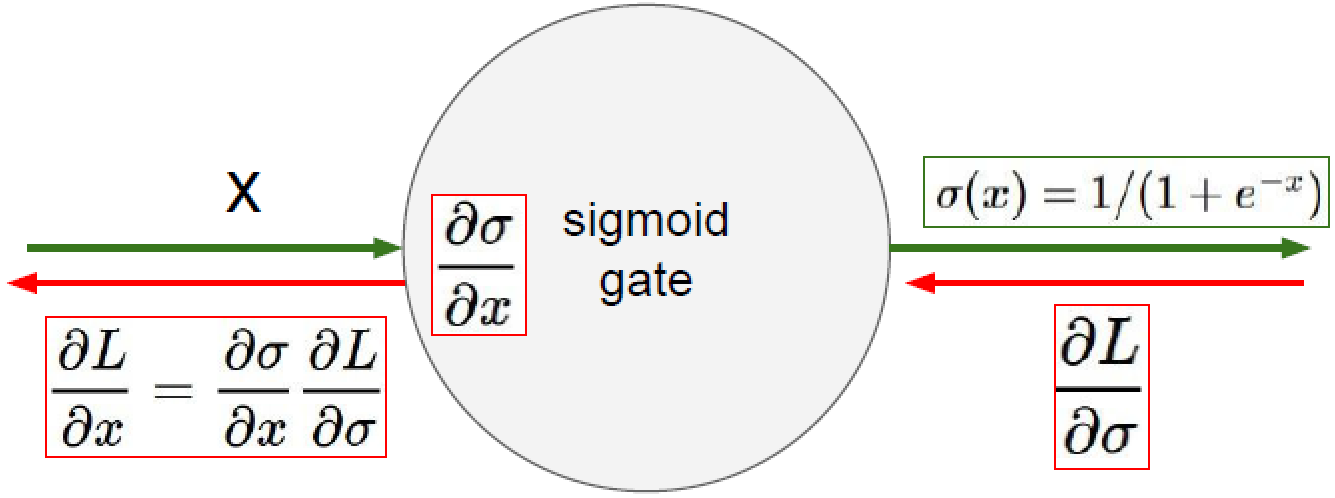
Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$



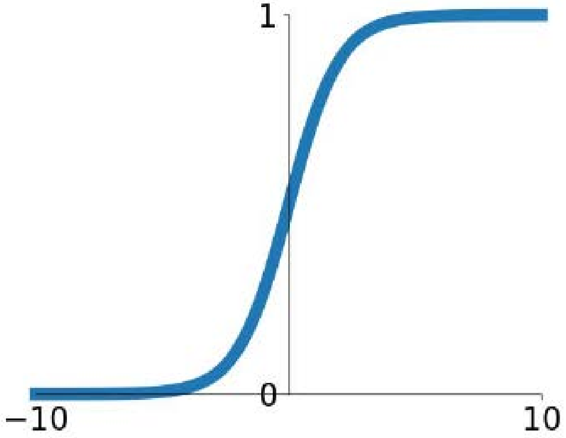
Problems:

1. The saturated neurons cause the **vanishing gradient** problem.
2. Sigmoid outputs are not zero-centered.
3. exp() is a bit computationally expensive

Vanishing Gradient Problem



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



[slide credit: Stanford CS231n]

All Positive Inputs

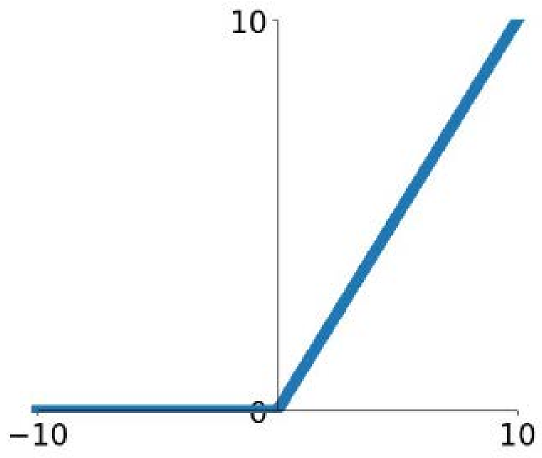
- What happens when the inputs to a layer are all positive?
- What can we say about the gradients with respect to $\mathbf{w}$?
- Suppose the loss is $L\big(f(\mathbf{x})\big)$ with $f(\mathbf{x}) = \sum_i w_i x_i + b$

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial \mathbf{w}} \qquad \frac{\partial f}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial f}{\partial w_1} & \cdots & \frac{\partial f}{\partial w_n} \end{bmatrix}^T = [x_1 \quad \cdots \quad x_n]^T$$

- Since $x_i > 0$, the gradient $\frac{\partial L}{\partial \mathbf{w}}$ always has the same sign as $\frac{\partial L}{\partial f}$ (all positive or all negative).

ReLU

ReLU: $\max(0, x)$



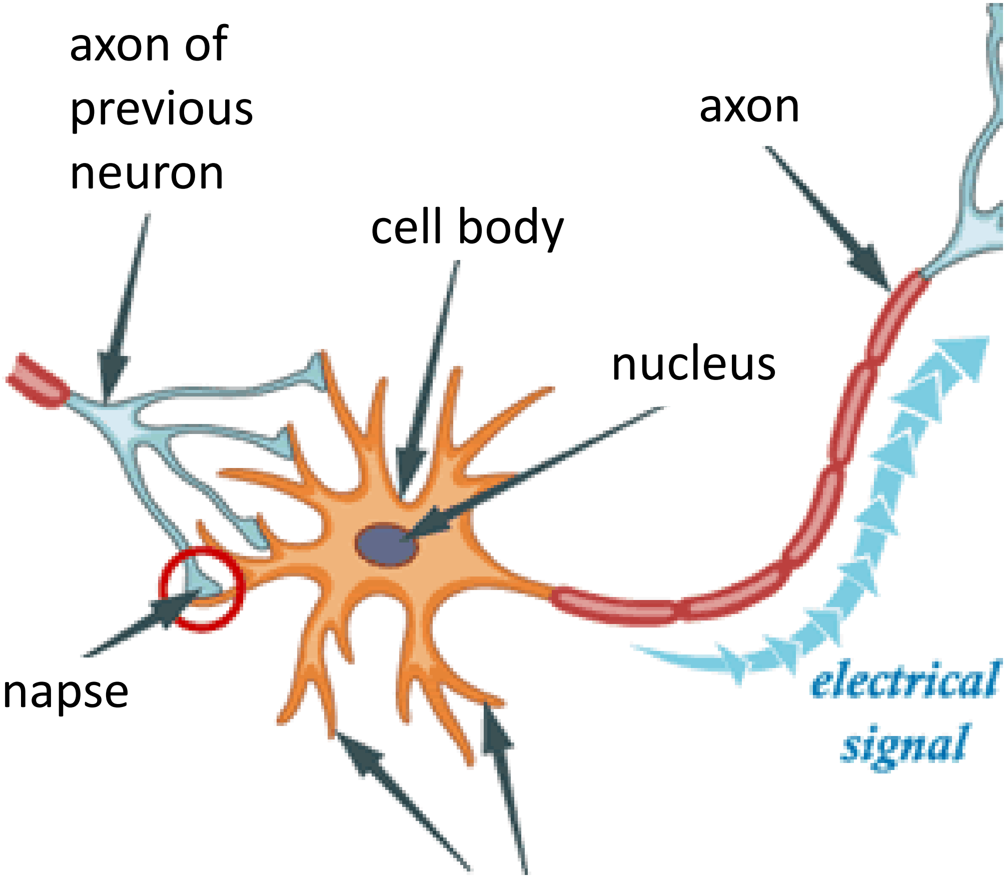
Pros:

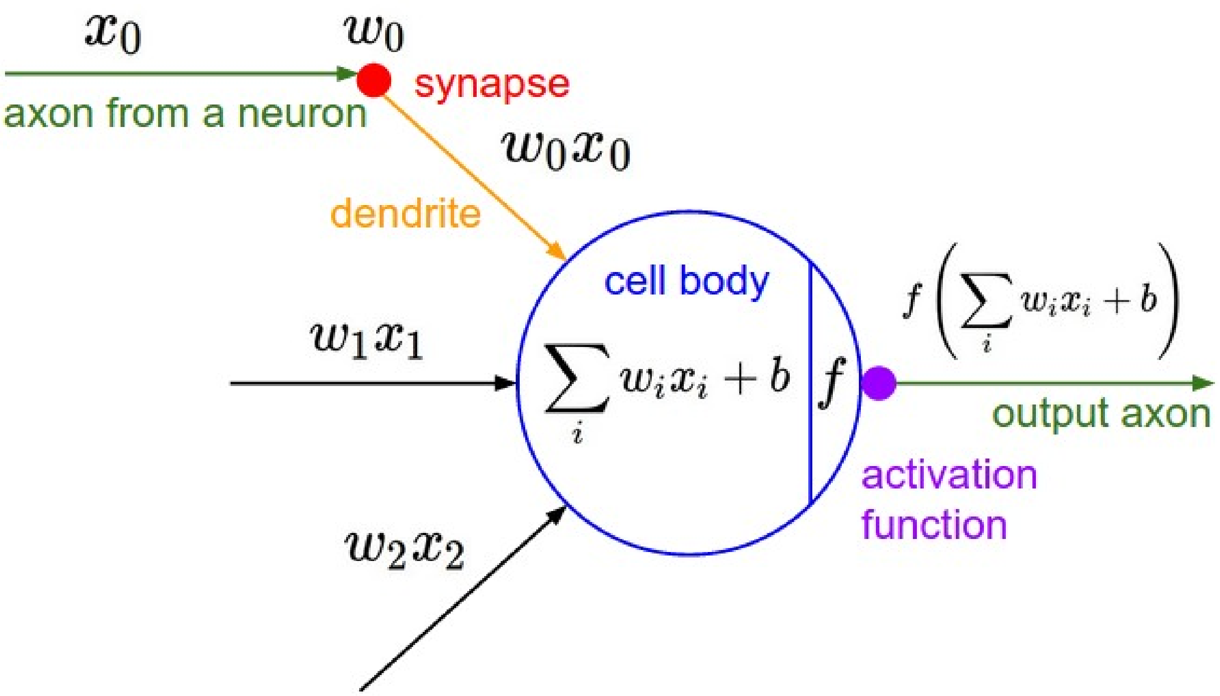
1. Does not saturate
2. Computationally efficient

Cons:

1. Not zero-entered output

Relation to Biological Neurons





Keras: model.summary()

```
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 3072)              0
_________________________________________________________________
dense (Dense)                (None, 512)               1573376
_________________________________________________________________
dense_1 (Dense)              (None, 128)               65664
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,640,330
Trainable params: 1,640,330
Non-trainable params: 0
```

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(32, 32, 3)),
    keras.layers.Dense(512, activation=tf.nn.relu),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)])
model.summary()
```

Keras: Training Pipeline

```
model = create_model()
model.compile(optimizer='adam',
          loss='sparse_categorical_crossentropy',
          metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=30)
```

Epoch 1/30 - 7s 140us/sample - loss: 1.8849 - acc: 0.3227
Epoch 2/30 - 7s 133us/sample - loss: 1.6746 - acc: 0.3999
Epoch 3/30 - 7s 131us/sample - loss: 1.5990 - acc: 0.4297
Epoch 4/30 - 7s 133us/sample - loss: 1.5544 - acc: 0.4458
Epoch 5/30 - 7s 131us/sample - loss: 1.5119 - acc: 0.4622
Epoch 6/30 - 7s 132us/sample - loss: 1.4852 - acc: 0.4726
Epoch 7/30 - 7s 132us/sample - loss: 1.4656 - acc: 0.4780
⋮
Epoch 28/30 - 7s 130us/sample - loss: 1.2196 - acc: 0.5629
Epoch 29/30 - 6s 130us/sample - loss: 1.2128 - acc: 0.5637
Epoch 30/30 - 6s 130us/sample - loss: 1.2103 - acc: 0.5634

Keras Callbacks

- model.fit(train_images, train_labels, epochs=30, callbacks = [cp_callback, tb_callback])

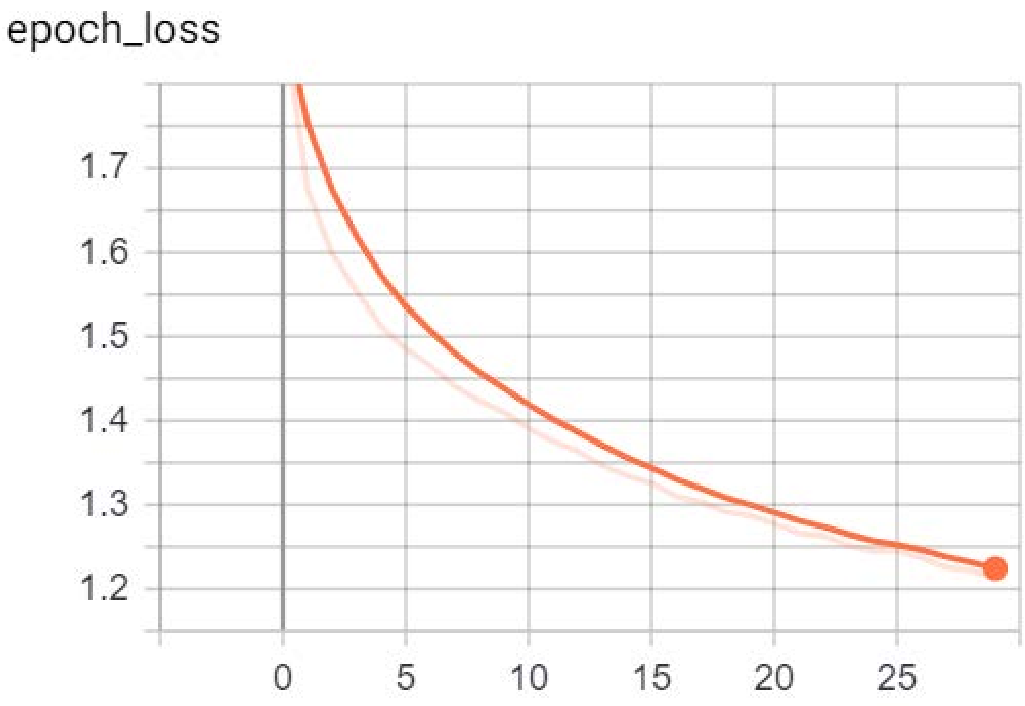
TensorBoard

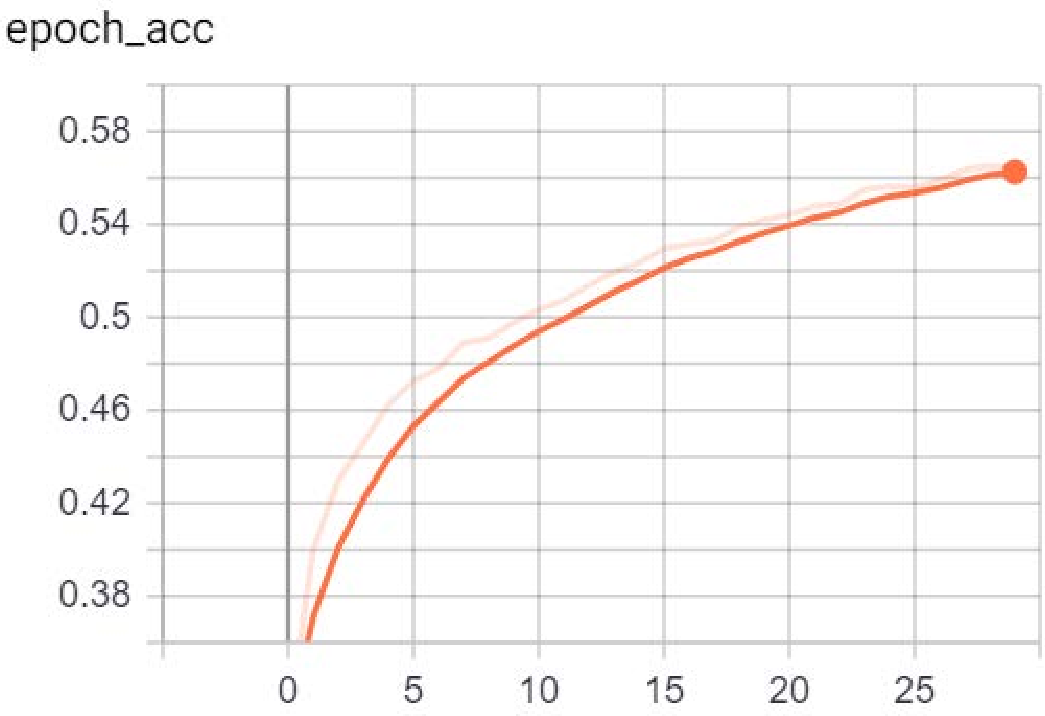
```
tb_callback = TensorBoard(log_dir='./logs',
             histogram_freq=0,
             write_graph=True,
             write_grads=True,
             write_images=True,
             embeddings_freq=0,
             embeddings_layer_names=None,
             embeddings_metadata=None)
```

Checkpoint

```
checkpoint_path = "training_cifar10/cp-{epoch:04d}.ckpt"
cp_callback = keras.callbacks.ModelCheckpoint(
                  checkpoint_path,
                  save_weights_only=True,
                  verbose=1,
                  period=10)
```

Keras: TensorBoard

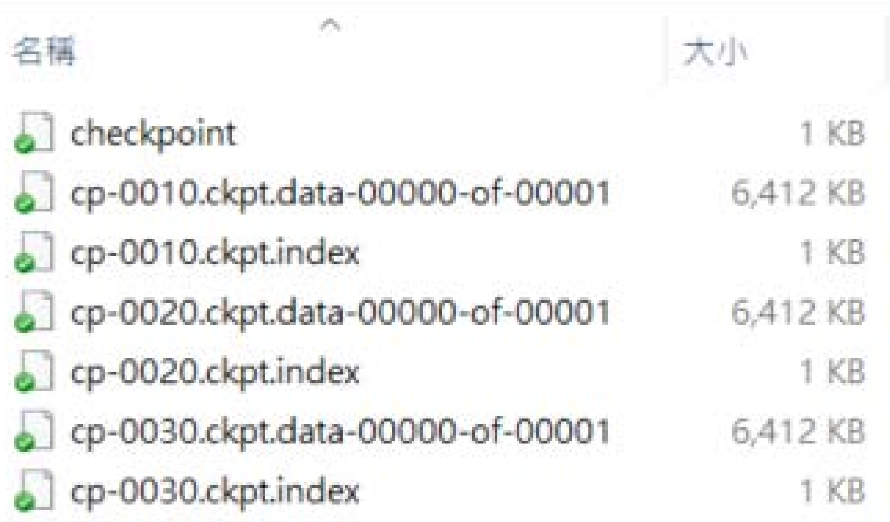




Keras: Checkpoint

```
checkpoint_path = "training_cifar10/cp-{epoch:04d}.ckpt"
loss, acc = model.evaluate(test_images, test_labels)

for epoch in [10, 20, 30]:
    latest = checkpoint_path.format(epoch=epoch)
    model.load_weights(latest)
    loss, acc = model.evaluate(test_images, test_labels)
```



名稱	大小
checkpoint	1 KB
cp-0010.ckpt.data-00000-of-00001	6,412 KB
cp-0010.ckpt.index	1 KB
cp-0020.ckpt.data-00000-of-00001	6,412 KB
cp-0020.ckpt.index	1 KB
cp-0030.ckpt.data-00000-of-00001	6,412 KB
cp-0030.ckpt.index	1 KB

Random Weights
1s 86us/sample - loss: 2.5282 - acc: 0.0875

Epoch 10 checkpoint
1s 60us/sample - loss: 1.5094 - acc: 0.4586

Epoch 20 checkpoint
1s 60us/sample - loss: 1.4365 - acc: 0.4936

Epoch 30 checkpoint
1s 59us/sample - loss: 1.4265 - acc: 0.5045