Neural Networks

CS5330, Huaizu Jiang

Fall 2021, Northeastern University

Recap

Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**

Classifier scores

$s = f(x_i, W)$

Softmax function

$$p_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)}$$

cat	**3.2**
car	5.1
frog	-1.7

Slide credit: D Fouhey & J Johnson

Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**

Classifier scores

$$s = f(x_i, W)$$

Softmax function

$$p_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)}$$

cat	**3.2**
car	5.1
frog	-1.7

Unnormalized log-probabilities / logits

Slide credit: D Fouhey & J Johnson

Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**

Classifier scores

$$s = f(x_i, W)$$

Softmax function

$$p_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)}$$

Probabilities must be >= 0

	Unnormalized log-probabilities / logits		unnormalized probabilities
cat	**3.2**	exp	**24.5**
car	5.1	→	164
frog	-1.7		0.18

Slide credit: D Fouhey & J Johnson

Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**

Classifier scores

$$s = f(x_i, W)$$

Softmax function

$$p_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)}$$

	Unnormalized log-probabilities / logits	exp	Probabilities must be >= 0 unnormalized probabilities	normalize	Probabilities must sum to 1 probabilities
cat	**3.2**		**24.5**		**0.13**
car	5.1		164		0.87
frog	-1.7		0.18		0.00

Slide credit: D Fouhey & J Johnson

Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**

Classifier scores

$$s = f(x_i, W)$$

Softmax function

$$p_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)}$$

Loss

$$L_i = -\log(p_{y_i})$$

	Unnormalized log-probabilities / logits	exp	Probabilities must be >= 0 unnormalized probabilities	normalize	Probabilities must sum to 1 probabilities
cat	**3.2**		**24.5**		**0.13**
car	5.1		164		0.87
frog	-1.7		0.18		0.00

$L_i = -\log(0.13)$

$= \mathbf{2.04}$

Slide credit: D Fouhey & J Johnson

Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**

Classifier scores

$$s = f(x_i, W)$$

Softmax function

$$p_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)}$$

Loss

$$L_i = -\log(p_{y_i})$$

	Unnormalized log-probabilities / logits	exp	Probabilities must be >= 0	normalize	Probabilities must sum to 1	probabilities
cat	**3.2**		**24.5**		**0.13**	
car	5.1		164		0.87	
frog	-1.7		0.18		0.00	

$L_i = -\log(0.13)$
$= \mathbf{2.04}$

Maximum Likelihood Estimation
Choose weights to maximize the likelihood of the observed data

Slide credit: D Fouhey & J Johnson

Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**

Classifier scores

$$s = f(x_i, W)$$

Softmax function

$$p_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)}$$

Loss

$$L_i = -\log(p_{y_i})$$

	Unnormalized log-probabilities / logits	exp	unnormalized probabilities	normalize	probabilities	Compare	Correct probs
	Probabilities must be >= 0				Probabilities must sum to 1		
cat	**3.2**		**24.5**		**0.13**		**1.00**
car	5.1		164		0.87		0.00
frog	-1.7		0.18		0.00		0.00

Slide credit: D Fouhey & J Johnson

Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**

Classifier scores

$$s = f(x_i, W)$$

Softmax function

$$p_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)}$$

Loss

$$L_i = -\log(p_{y_i})$$

	Unnormalized log-probabilities / logits	exp	unnormalized probabilities	normalize	probabilities	Compare	Correct probs
	Probabilities must be >= 0				Probabilities must sum to 1		
cat	**3.2**		**24.5**		**0.13**		**1.00**
car	5.1		164		0.87		0.00
frog	-1.7		0.18		0.00		0.00
					Q		P

Cross-Entropy:

$$H(P, Q) = -\sum_y P(y) \log Q(y)$$

Slide credit: D Fouhey & J Johnson

Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**

Classifier scores

$$s = f(x_i, W)$$

Softmax function

$$p_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)}$$

Loss

$$L_i = -\log(p_{y_i})$$

cat	**3.2**
car	5.1
frog	-1.7

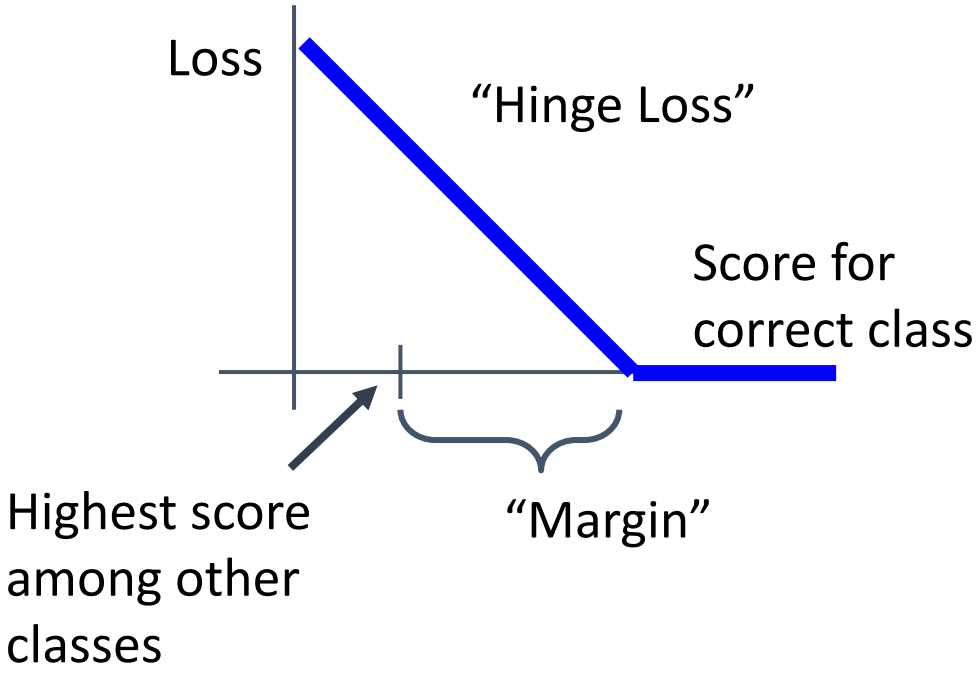
Putting it all together:

$$L_i = -\log \frac{\exp(s_{y_i})}{\sum_j \exp(s_j)}$$

Slide credit: D Fouhey & J Johnson

Multiclass SVM Loss

"The score of the correct class should be higher than all the other scores"



Given an example (x_i, y_i)
(x_i is image, y_i is label)

Let $s = f(x_i, W)$ be scores

Then the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

margin

Slide credit: D Fouhey & J Johnson

Multiclass SVM Loss



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Loss	2.9		

Given an example (x_i, y_i)
(x_i is image, y_i is label)

Let $s = f(x_i, W)$ be scores

Then the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

= max(0, 5.1 - 3.2 + 1)
+ max(0, -1.7 - 3.2 + 1)
= max(0, 2.9) + max(0, -3.9)
= 2.9 + 0
= 2.9

Slide credit: D Fouhey & J Johnson

Cross-Entropy vs SVM Loss

$$L_i = -\log \frac{\exp(s_{y_i})}{\sum_j \exp(s_j)}$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Assume scores:

[10, -2, 3]

[10, 9, 9]

[10, -100, -100]

and $y_i = 0$

Q: What is cross-entropy loss? What is SVM loss?

A: Cross-entropy loss > 0
SVM loss = 0

Slide credit: D Fouhey & J Johnson

Cross-Entropy vs SVM Loss

$$L_i = -\log \frac{\exp(s_{y_i})}{\sum_j \exp(s_j)}$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Assume scores:

[10, -2, 3]

[10, 9, 9]

[10, -100, -100]

and $y_i = 0$

Q: What happens to each loss if I slightly change the scores of the last datapoint?

A: Cross-entropy loss will change; SVM loss will stay the same

Slide credit: D Fouhey & J Johnson

Cross-Entropy vs SVM Loss

$$L_i = -\log \frac{\exp(s_{y_i})}{\sum_j \exp(s_j)}$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Assume scores:

[20, -2, 3]

[20, 9, 9]

[20, -100, -100]

and $y_i = 0$

Q: What happens to each loss if I double the score of the correct class from 10 to 20?

A: Cross-entropy loss will decrease, SVM loss still 0

Slide credit: D Fouhey & J Johnson

There is a “bug” with the hinge loss:

$$f(x, W) = Wx$$

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$$

E.g. Suppose that we found a W such that L = 0. Is this W unique?

$$f(x, W) = Wx$$

cat	1.3	2.6
car	**2.5**	**5.0**
frog	2.0	4.0
Loss:	0.5	0

An example:

What is the loss?

How could we change W to eliminate the loss? (POLL)

Multiply W (and b) by 2!

Wait a minute! Have we done anything useful???

No! Any example that used to be wrong is still wrong (on the wrong side of the boundary). Any example that is right is still right (on the correct side of the boundary).

Slide credit: E Learned-Miller, originally developed by A Karpathy, J Johnson, Fei-Fei L

Regularization

λ = regularization strength (hyperparameter)

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from having too much flexibility.

Simple examples

L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Slide credit: E Learned-Miller, originally developed by A Karpathy, J Johnson, Fei-Fei L

Optimization

Goal: find the **w** minimizing some loss function L.

$$\arg\min_{\boldsymbol{w}\in R^D} L(\boldsymbol{w})$$

Works for lots of different Ls:

$$L(\boldsymbol{W}) = \boldsymbol{\lambda}\|\boldsymbol{W}\|_{\boldsymbol{2}}^{\boldsymbol{2}} + \sum_{i=1}^{n} -\log\left(\frac{\exp((Wx)_{y_i})}{\sum_k \exp((Wx)_k))}\right)$$

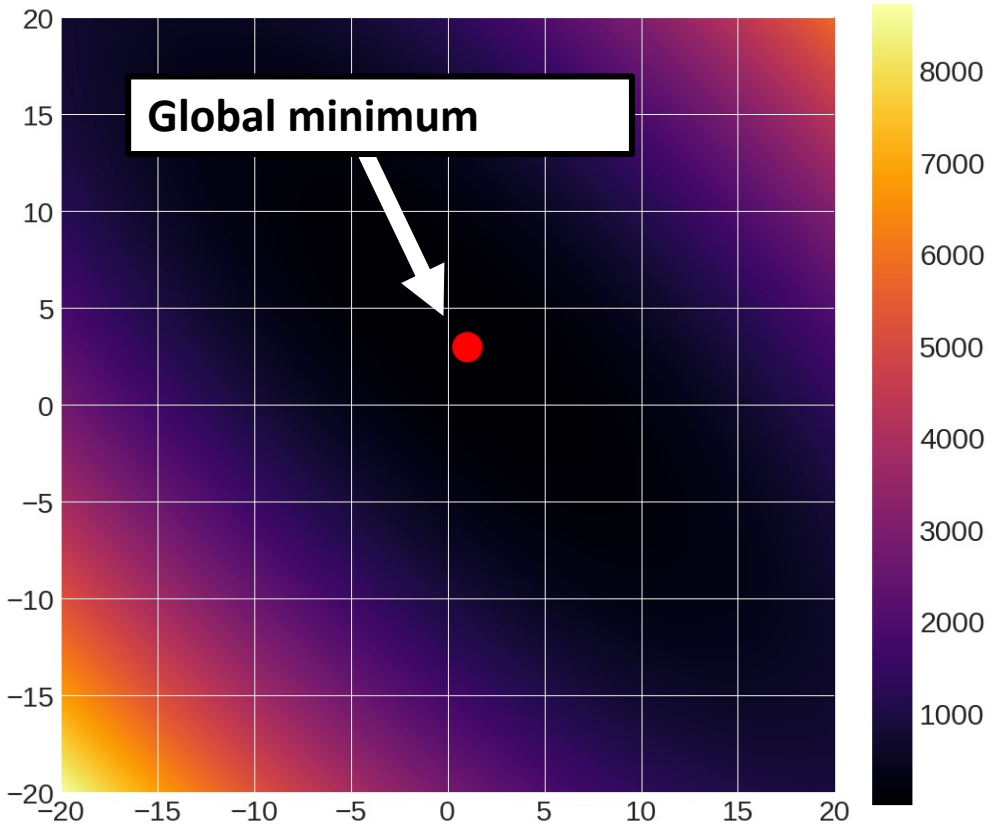
$$L(\boldsymbol{w}) = C\|\boldsymbol{w}\|_2^2 + \sum_{i=1}^{n} \max(0, 1 - y_i \boldsymbol{w}^T \boldsymbol{x_i})$$

$$L(\boldsymbol{w}) = \lambda\|\boldsymbol{w}\|_2^2 + \sum_{i=1}^{n} \left(y_i - \boldsymbol{w^T x_i}\right)^2$$

Slide credit: D Fouhey & J Johnson

Sample Function to Optimize

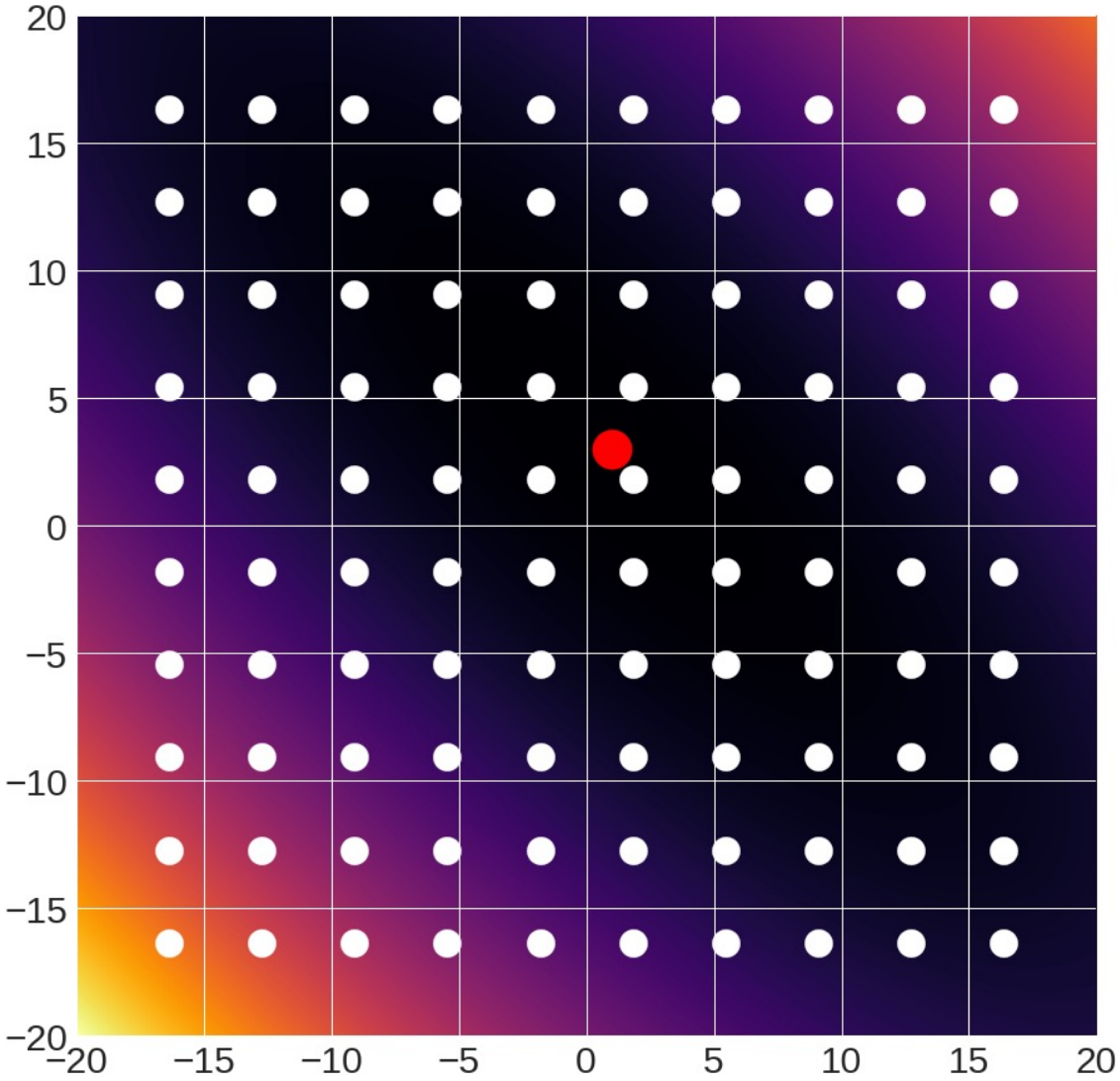
$$f(x,y) = (x+2y-7)^2 + (2x+y-5)^2$$



Warning: This is 2D, intuition may not generalize to high dimension

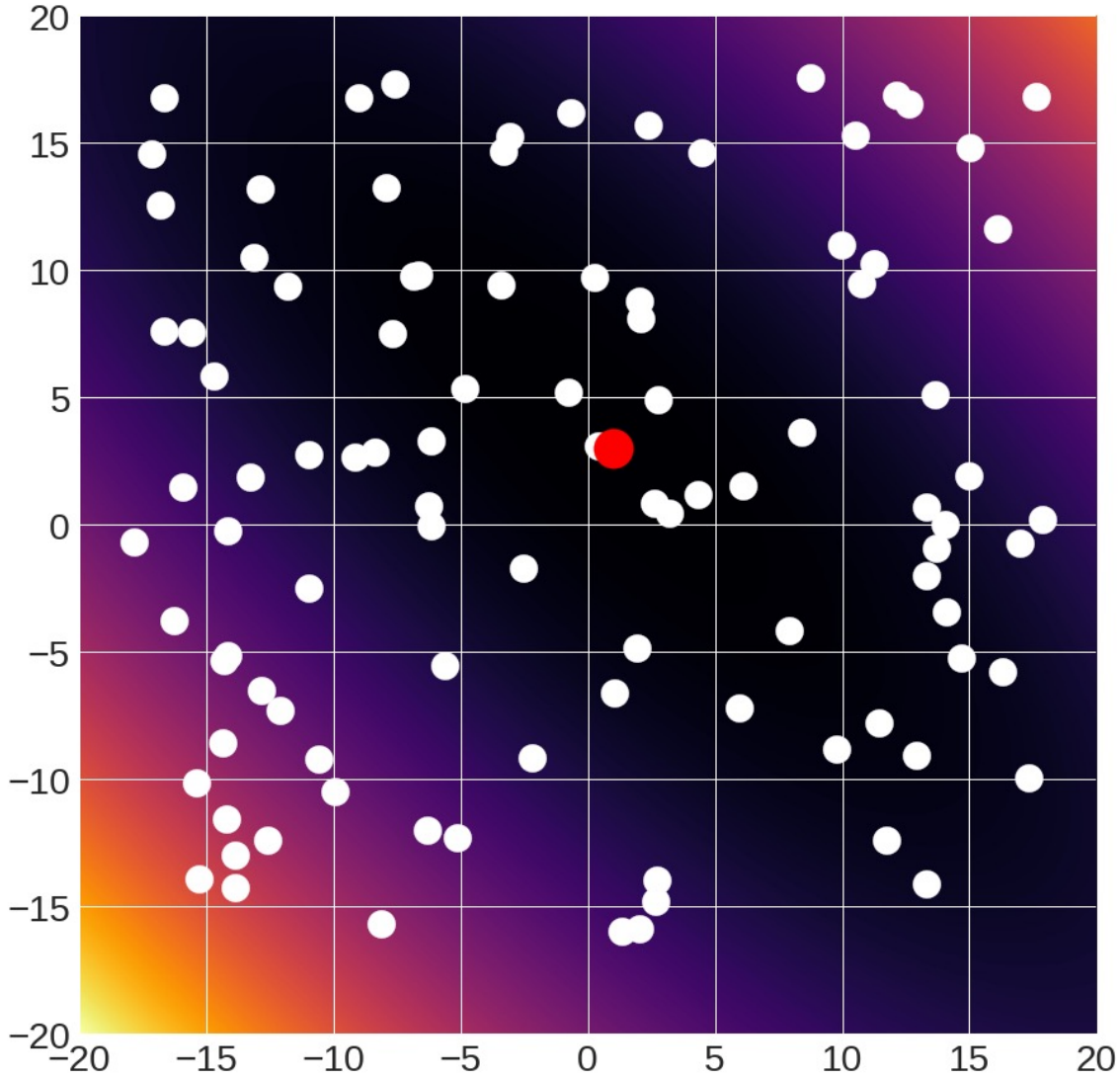
Slide credit: D Fouhey & J Johnson

Idea #1A: Grid Search



Slide credit: D Fouhey & J Johnson

Option #1B: Random Search



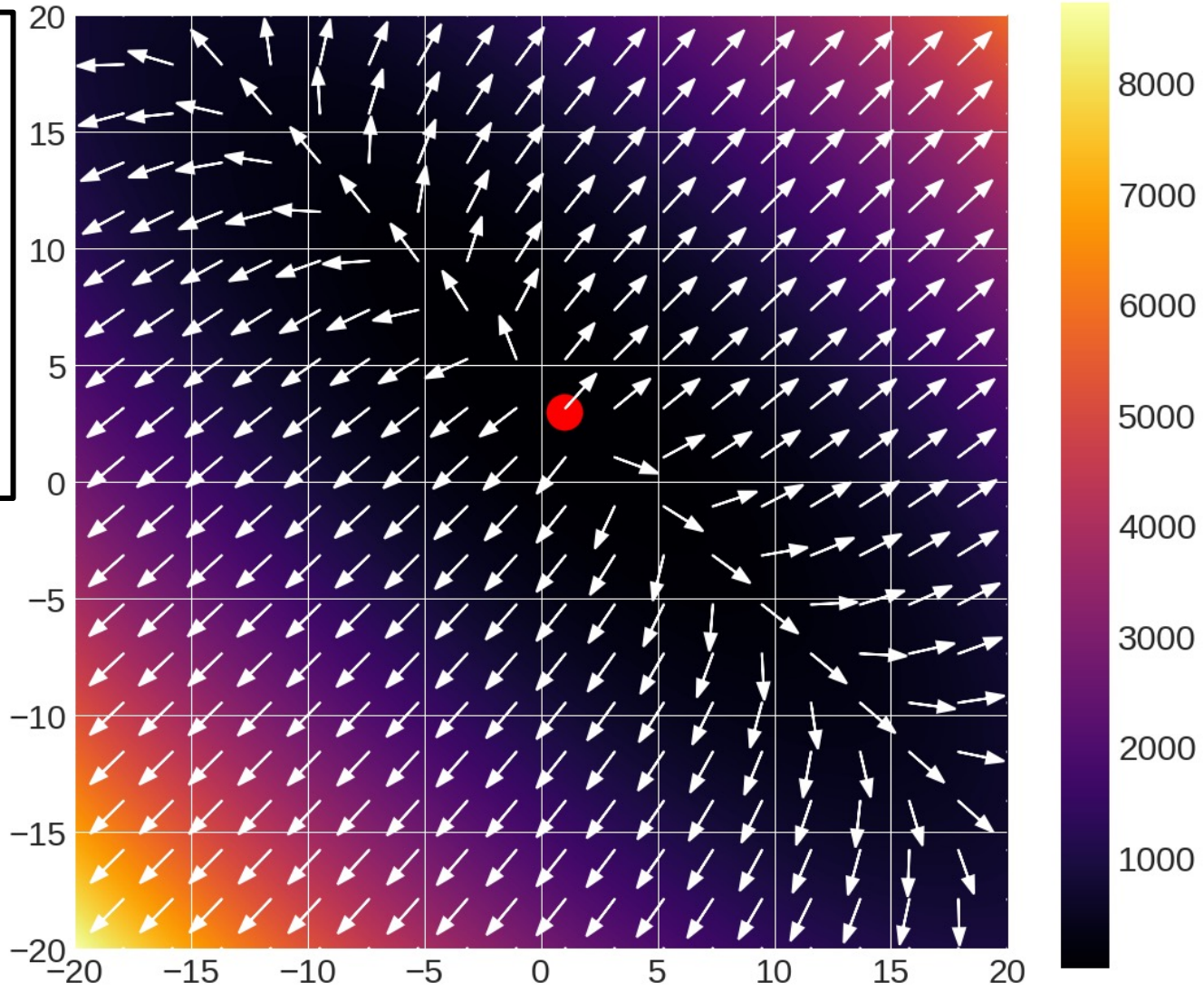
Slide credit: D Fouhey & J Johnson

Idea #2: Follow the slope

Slide credit: E Learned-Miller, originally developed by A Karpathy, J Johnson, Fei-Fei L

Idea #2: Follow the slope

Arrows:
gradient direction
(scaled to unit length)



Slide credit: D Fouhey & J Johnson

Idea #2: Follow the slope

Method: at each step, move in direction of negative gradient

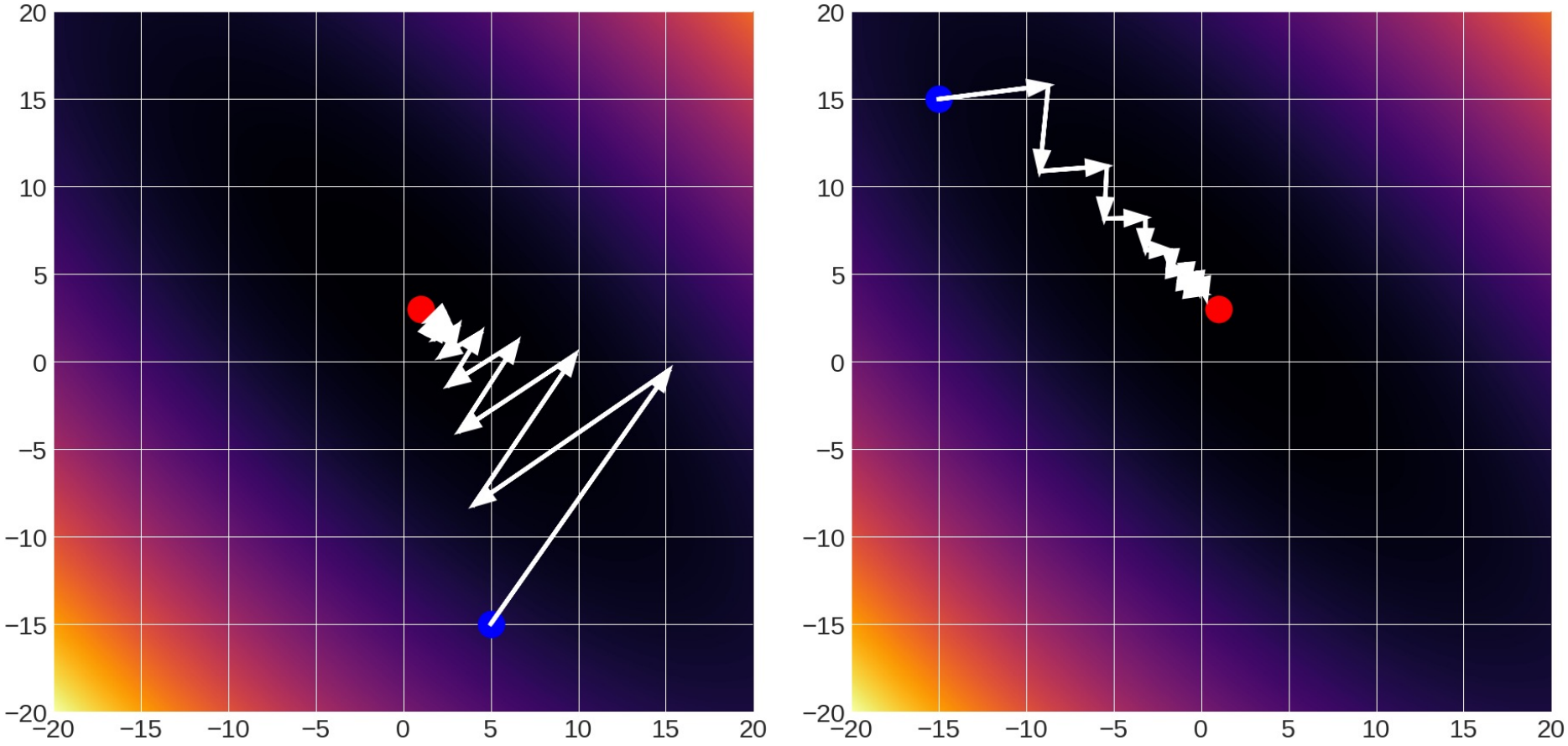
w0 = *initialize*() #initialize
for iter in range(numIters):
    $\mathbf{g} = \nabla_{\boldsymbol{w}} L(\boldsymbol{w})$ # eval gradient
    w = **w** + -*stepsize****g** # update w
return **w**

Slide credit: D Fouhey & J Johnson

Gradient Descent

Given starting point (blue)

$w_{i+1} = w_i + -9.8x10^{-2} * \text{gradient}$



Slide credit: D Fouhey & J Johnson

Computing Gradients: Numeric

How Do You Compute The Gradient?

Numerical Method:

$$\nabla_{\boldsymbol{w}} L(\boldsymbol{w}) = \begin{bmatrix} \frac{\partial L(w)}{\partial w_1} \\ \vdots \\ \frac{\partial L(w)}{\partial w_n} \end{bmatrix}$$

How do you compute this?

$$\frac{\partial f(x)}{\partial x} = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

In practice, use:

$$\frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon}$$

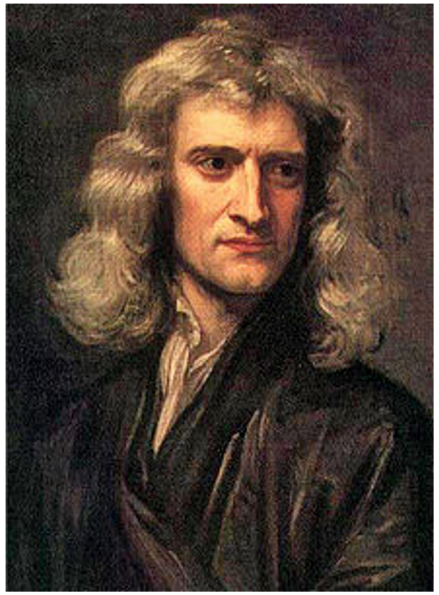
Slide credit: D Fouhey & J Johnson

Computing Gradients: Analytic

How Do You Compute The Gradient?

$$\nabla_{\boldsymbol{w}} L(\boldsymbol{w}) = \begin{bmatrix} \frac{\partial L(w)}{\partial x_1} \\ \vdots \\ \frac{\partial L(w)}{\partial x_n} \end{bmatrix}$$

Better Idea: Use Calculus!



This image is in the public domain

This image is in the public domain

Slide credit: D Fouhey & J Johnson

Computing Gradients

- **Numeric gradient**: approximate, slow, easy to write
- **Analytic gradient**: exact, fast, error-prone

In practice: Always use analytic gradient, but check implementation with numerical gradient. This is called a **gradient check.**

`torch.autograd.gradcheck(`*`func`*`,` *`inputs`*`,` *`eps=1e-06`*`,` *`atol=1e-05`*`,` *`rtol=0.001`*`,` *`raise_exception=True`*`,` *`check_sparse_nnz=False`*`,` *`nondet_tol=0.0`*`)` [SOURCE]

Check gradients computed via small finite differences against analytical gradients w.r.t. tensors in `inputs` that are of floating point type and with `requires_grad=True`.

The check between numerical and analytical gradients uses `allclose()`.

Slide credit: D Fouhey & J Johnson

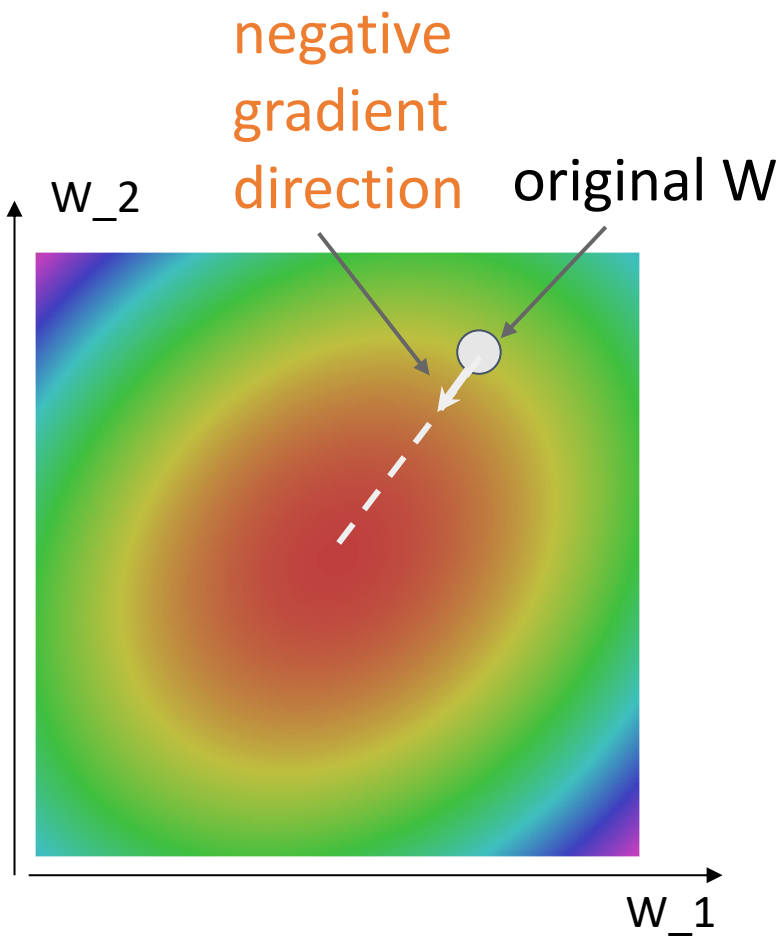
Gradient Descent

Iteratively step in the direction of the negative gradient
(direction of local steepest descent)

```
# Vanilla gradient descent
w = initialize_weights()
for t in range(num_steps):
  dw = compute_gradient(loss_fn, data, w)
  w -= learning_rate * dw
```

Hyperparameters:

- Weight initialization method
- Number of steps
- Learning rate



Slide credit: D Fouhey & J Johnson

Batch Gradient Descent

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

Problem: Full sum is expensive when N is large!

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

Slide credit: D Fouhey & J Johnson

Batch Gradient Descent

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

Problem: Full sum is expensive when N is large!

$$\nabla_W L(W) = \frac{1}{\boldsymbol{B}}\sum_{i=1}^{\boldsymbol{B}} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

Solution: Approximate sum using a **minibatch** of examples, e.g. *B*=32

Slide credit: D Fouhey & J Johnson

Stochastic Gradient Descent (SGD)

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

Problem: Full sum is expensive when N is large!

$$\nabla_W L(W) = \frac{1}{\boldsymbol{B}}\sum_{i=1}^{\boldsymbol{B}} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

Solution: Approximate sum using a **minibatch** of examples, e.g. *B*=32

```
# Stochastic gradient descent
w = initialize_weights()
for t in range(num_steps):
  minibatch = sample_data(data, batch_size)
  dw = compute_gradient(loss_fn, minibatch, w)
  w -= learning_rate * dw
```

Hyperparameters:

- Weight initialization
- Number of steps
- Learning rate
- Batch size
- Data sampling

Slide credit: D Fouhey & J Johnson

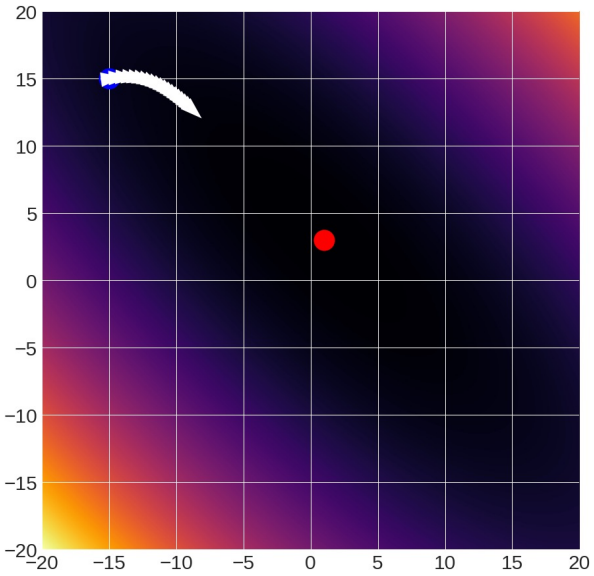
Gradient Descent: Learning Rate

Step size (also called **learning rate / lr**)

critical parameter

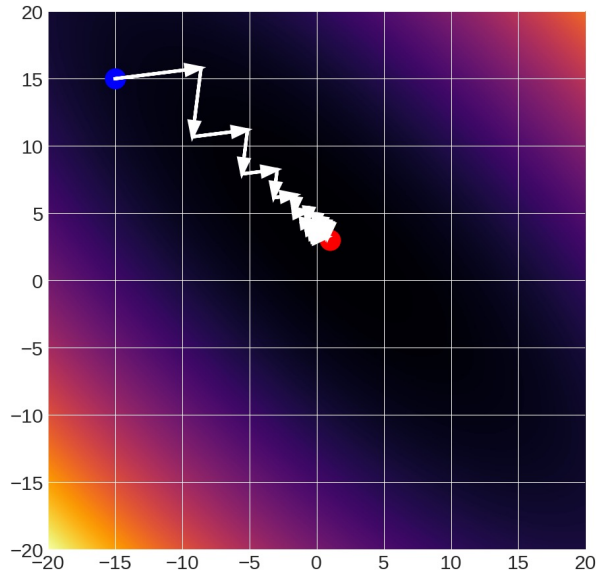
$1x10^{-2}$

falls short



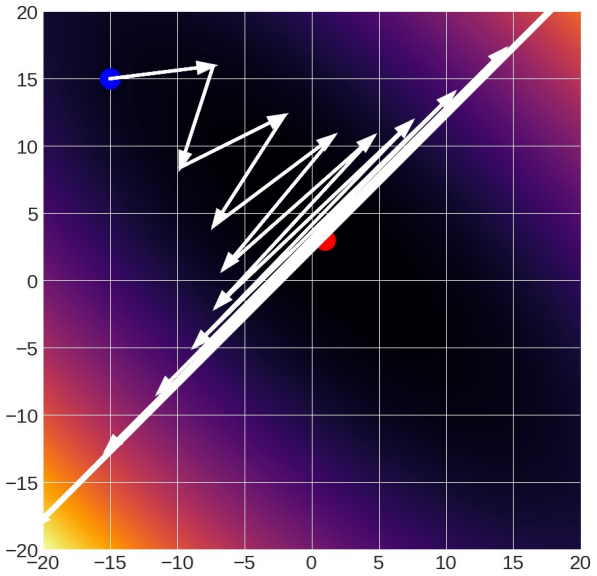
$10x10^{-2}$

converges



$12x10^{-2}$

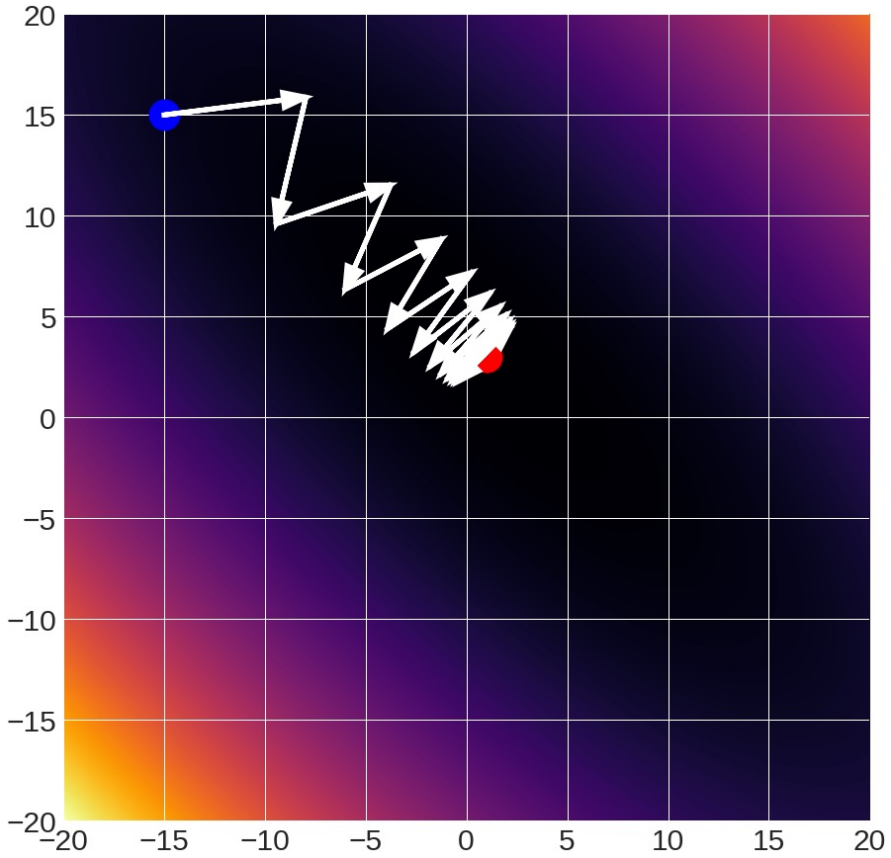
diverges



Slide credit: D Fouhey & J Johnson

Gradient Descent: Learning Rate

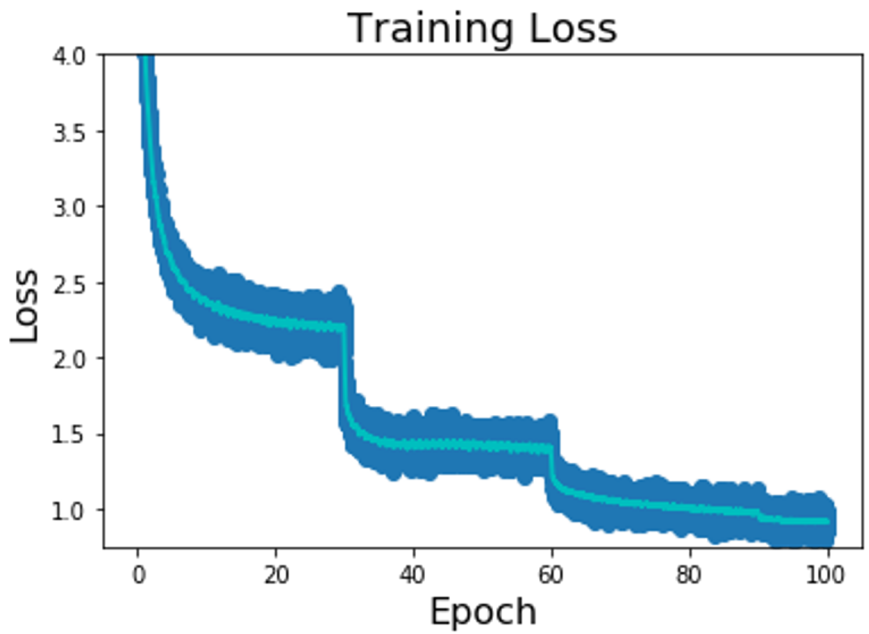
$11x10^{-2}$:oscillates

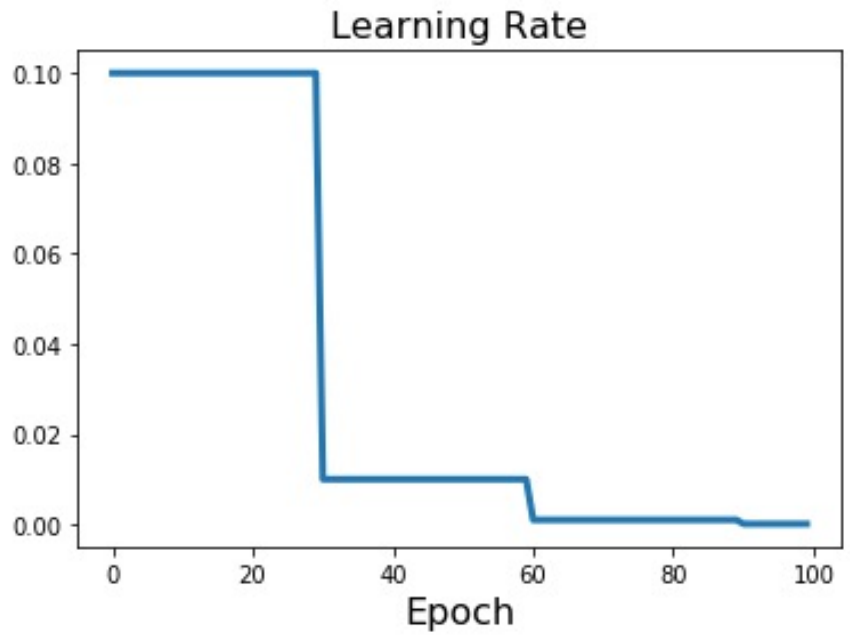


Slide credit: D Fouhey & J Johnson

Learning Rate Decay

Idea: Start with high learning rate, reduce it over time.
Step Decay: Reduce by some factor at fixed iterations



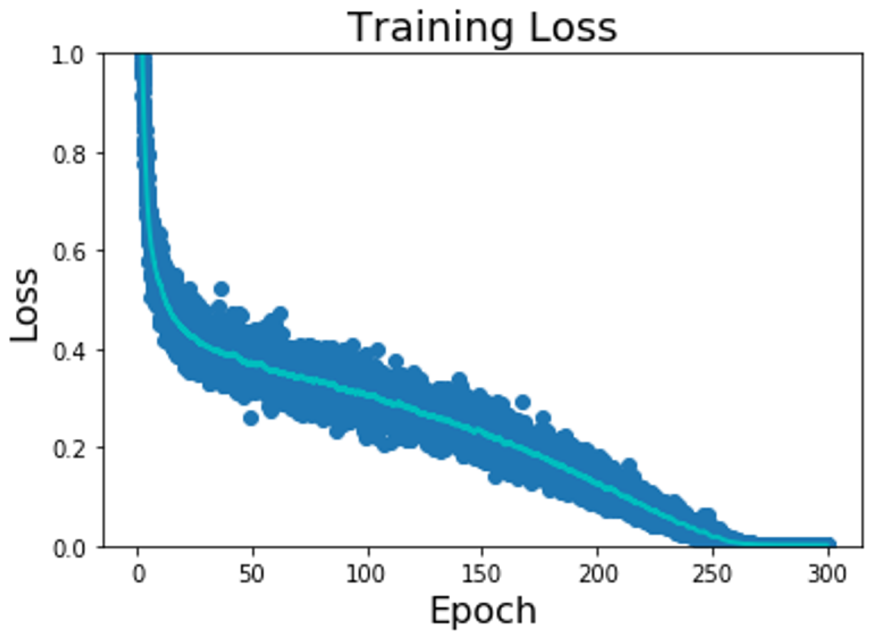


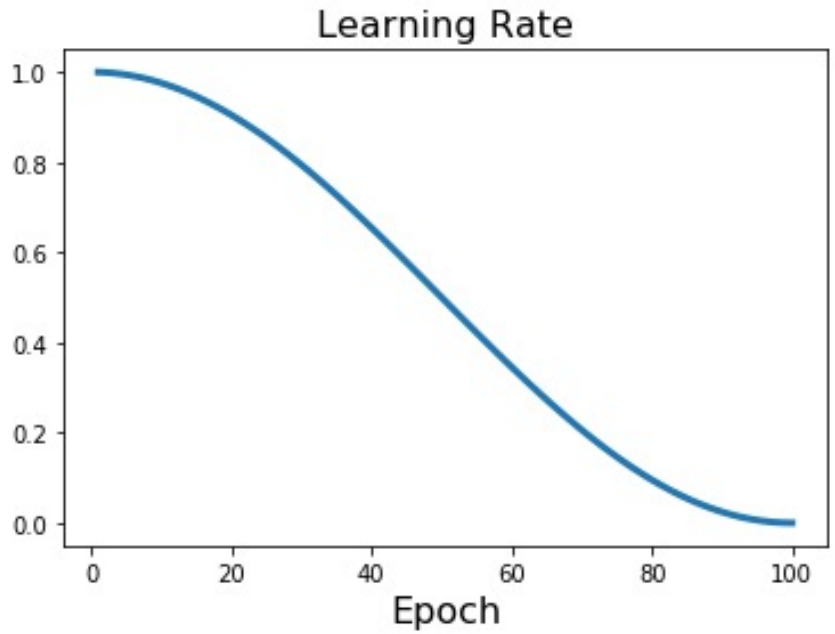
Slide credit: D Fouhey & J Johnson

Learning Rate Decay

Idea: Start with high learning rate, reduce it over time.

Cosine Decay: $\alpha_t = \frac{1}{2}\alpha_0\left(1 + cos\left(\frac{t\pi}{T}\right)\right)$





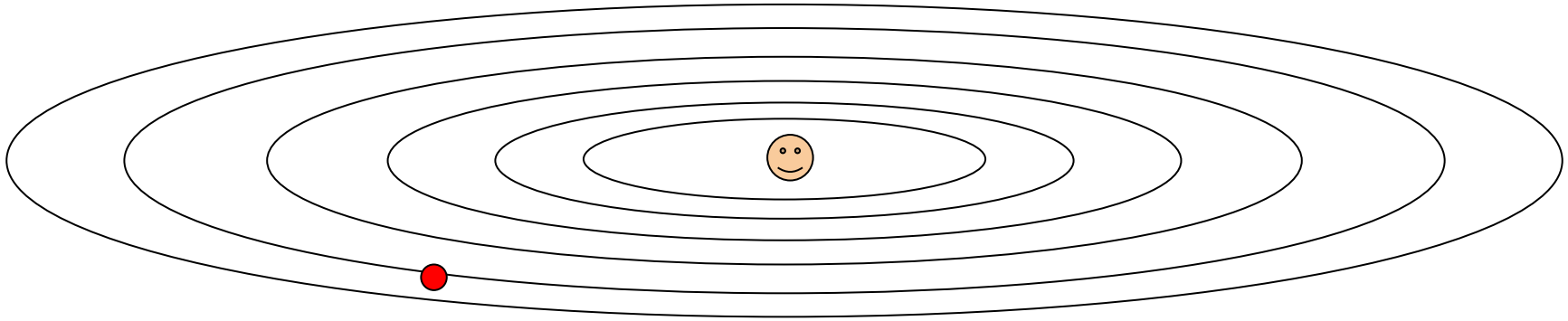
Slide credit: D Fouhey & J Johnson

Today's Class

- A bit more about SGD
- Neural Networks

Problems with SGD

What if loss changes quickly in one direction and slowly in another?



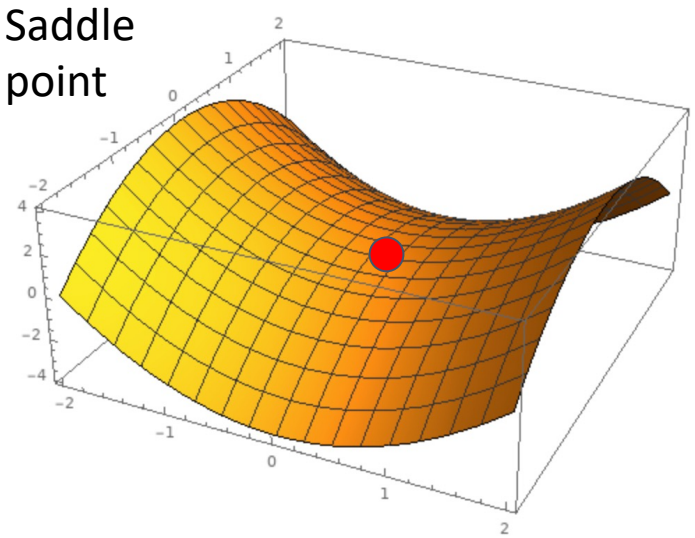
Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

Slide credit: D Fouhey & J Johnson

Problems with SGD

What if the loss function has a **local minimum** or **saddle point**?

Gradient is zero,
SGD gets stuck



Slide credit: D Fouhey & J Johnson

Problems with SGD

Our gradients come from minibatches so they can be noisy!

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

Slide credit: D Fouhey & J Johnson

SGD

SGD

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

```
for t in range(num_steps):
  dw = compute_gradient(w)
  w -= learning_rate * dw
```

Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

Slide credit: D Fouhey & J Johnson

SGD + Momentum

SGD

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

```
for t in range(num_steps):
  dw = compute_gradient(w)
  w -= learning_rate * dw
```

SGD + Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
v = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  v = rho * v + dw
  w -= learning_rate * v
```

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically $\rho = 0.9$ or 0.99

Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

Slide credit: D Fouhey & J Johnson

SGD + Momentum

SGD + Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t)$$
$$x_{t+1} = x_t + v_{t+1}$$

```
v = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  v = rho * v - learning_rate * dw
  w += v
```

SGD + Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
v = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  v = rho * v + dw
  w -= learning_rate * v
```

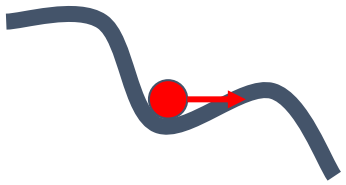
You may see SGD+Momentum formulated different ways, but they are equivalent - give same sequence of x

Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

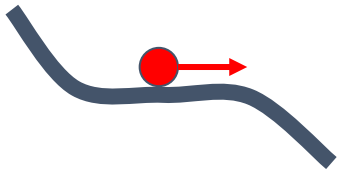
Slide credit: D Fouhey & J Johnson

SGD + Momentum

Local Minima



Saddle points



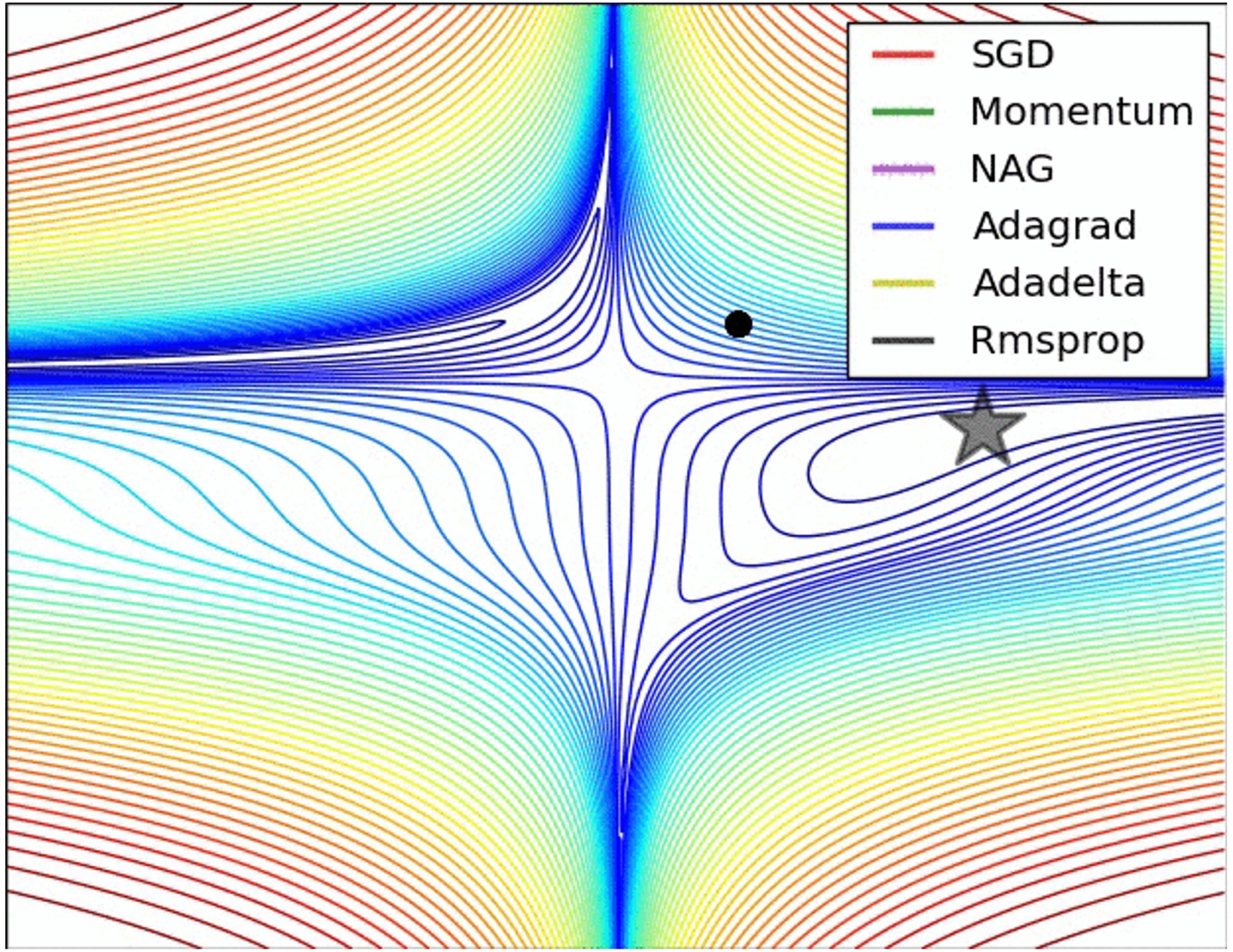
Gradient Noise

SGD SGD+Momentum

Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

Slide credit: D Fouhey & J Johnson

The effects of different update form formulas



(image credits to Alec Radford)

Slide credit: E Learned-Miller

Optimization in Practice

- **Conventional wisdom**: minibatch stochastic gradient descent (SGD) + momentum (package implements it for you) + some sensibly changing learning rate
- The above is typically what is meant by “SGD”
- Other update rules exist (Adam very common); sometimes better, sometimes worse than SGD

Slide credit: D Fouhey & J Johnson

Optimizing Everything

$$L(\boldsymbol{W}) = \boldsymbol{\lambda}\|\boldsymbol{W}\|_{\boldsymbol{2}}^{\boldsymbol{2}} + \sum_{i=1}^{n} -\log\left(\frac{\exp((Wx)_{y_i})}{\sum_k \exp((Wx)_k))}\right)$$

$$L(\boldsymbol{w}) = \lambda\|\boldsymbol{w}\|_2^2 + \sum_{i=1}^{n}\left(y_i - \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}_{\boldsymbol{i}}\right)^2$$

- (Automatically) Optimize **w** on training set with SGD to maximize training accuracy
- (Manually) Optimize λ with random/grid search to maximize validation accuracy

Slide credit: D Fouhey & J Johnson

Where we are:

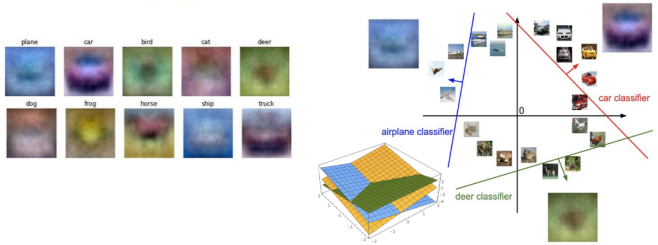
1. Use **Linear Models** for image classification problems

2. Use **Loss Functions** to express preferences over different choices of weights

3. Use **Stochastic Gradient Descent** to minimize our loss functions and train the model

4. Add **Regularization** to control overfitting

$$s = f(x; W) = Wx$$



$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$ Softmax

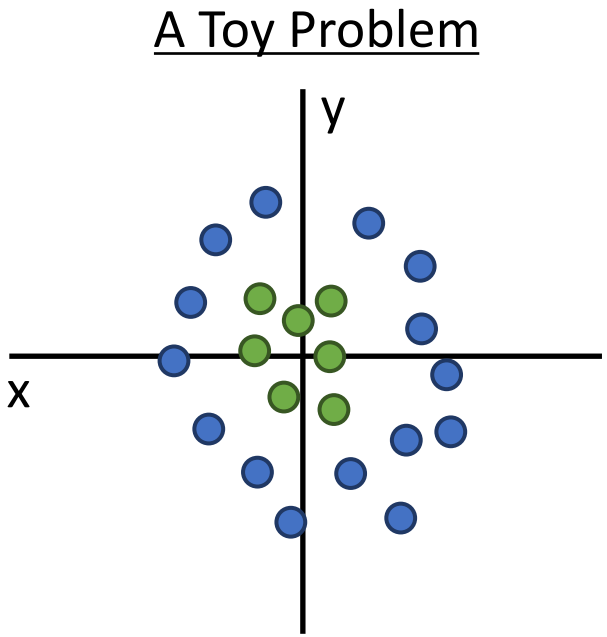
$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$ SVM

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$

```
v = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  v = rho * v + dw
  w -= learning_rate * v
```

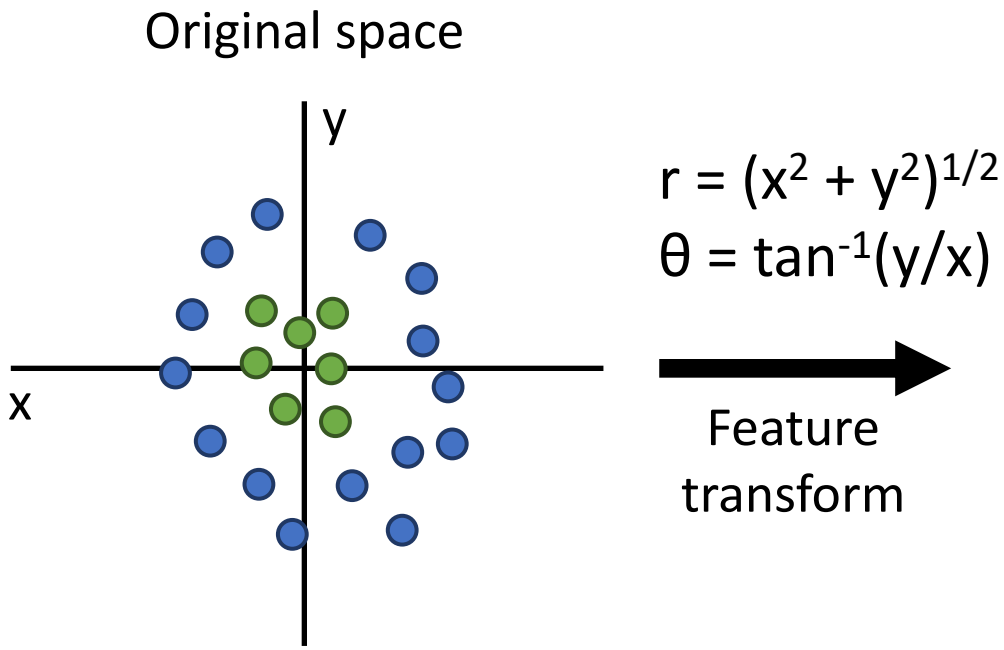
Slide credit: D Fouhey & J Johnson

Problem: Linear Classifiers not enough



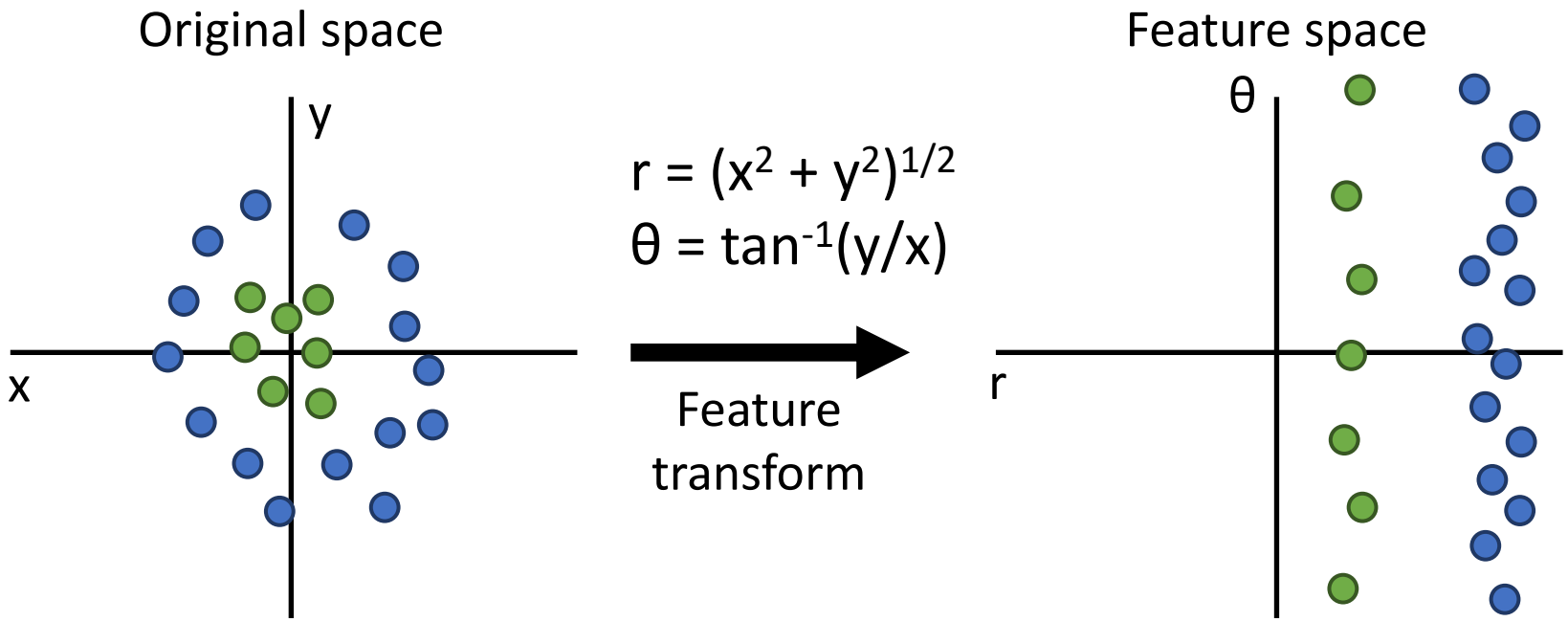
Slide credit: D Fouhey & J Johnson

One solution: **Feature Transforms**



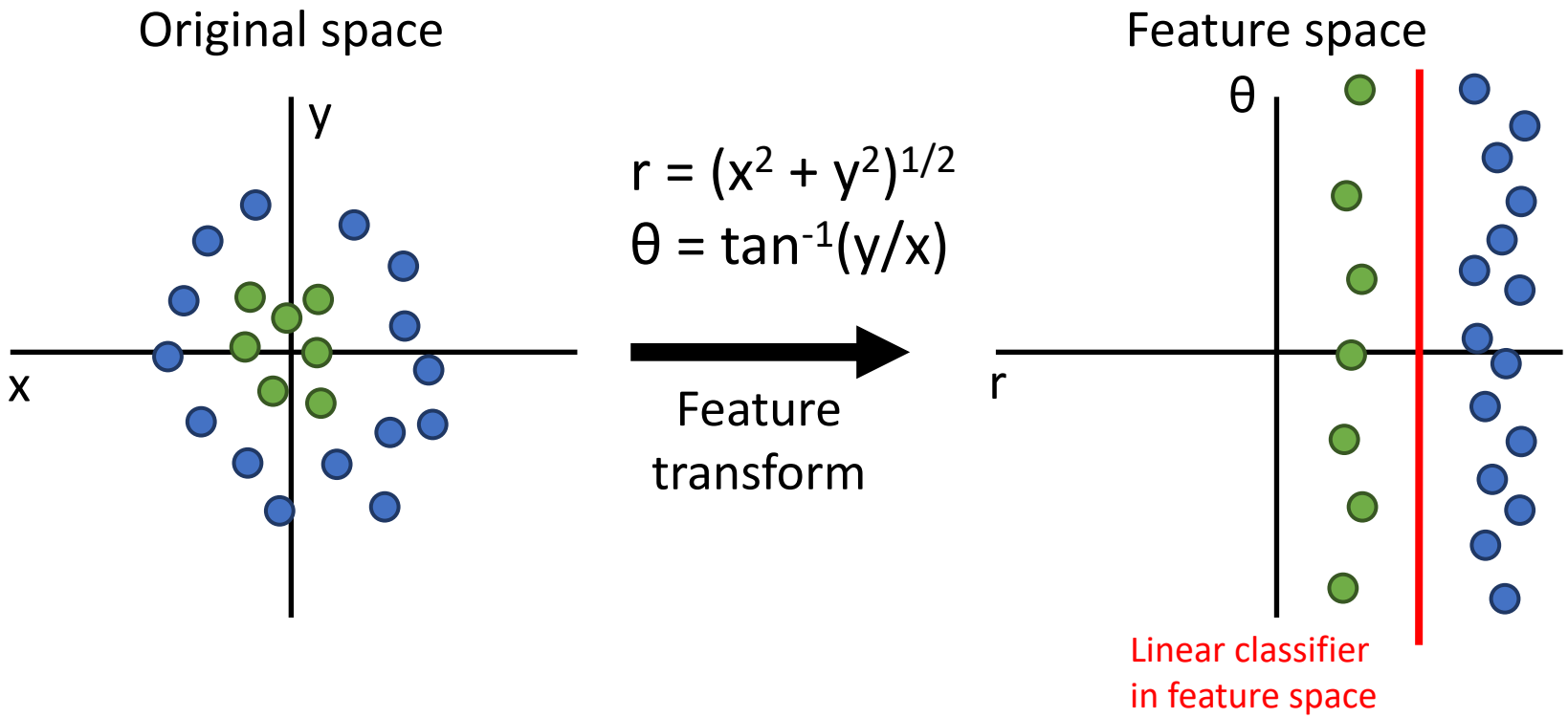
Slide credit: D Fouhey & J Johnson

One solution: **Feature Transforms**



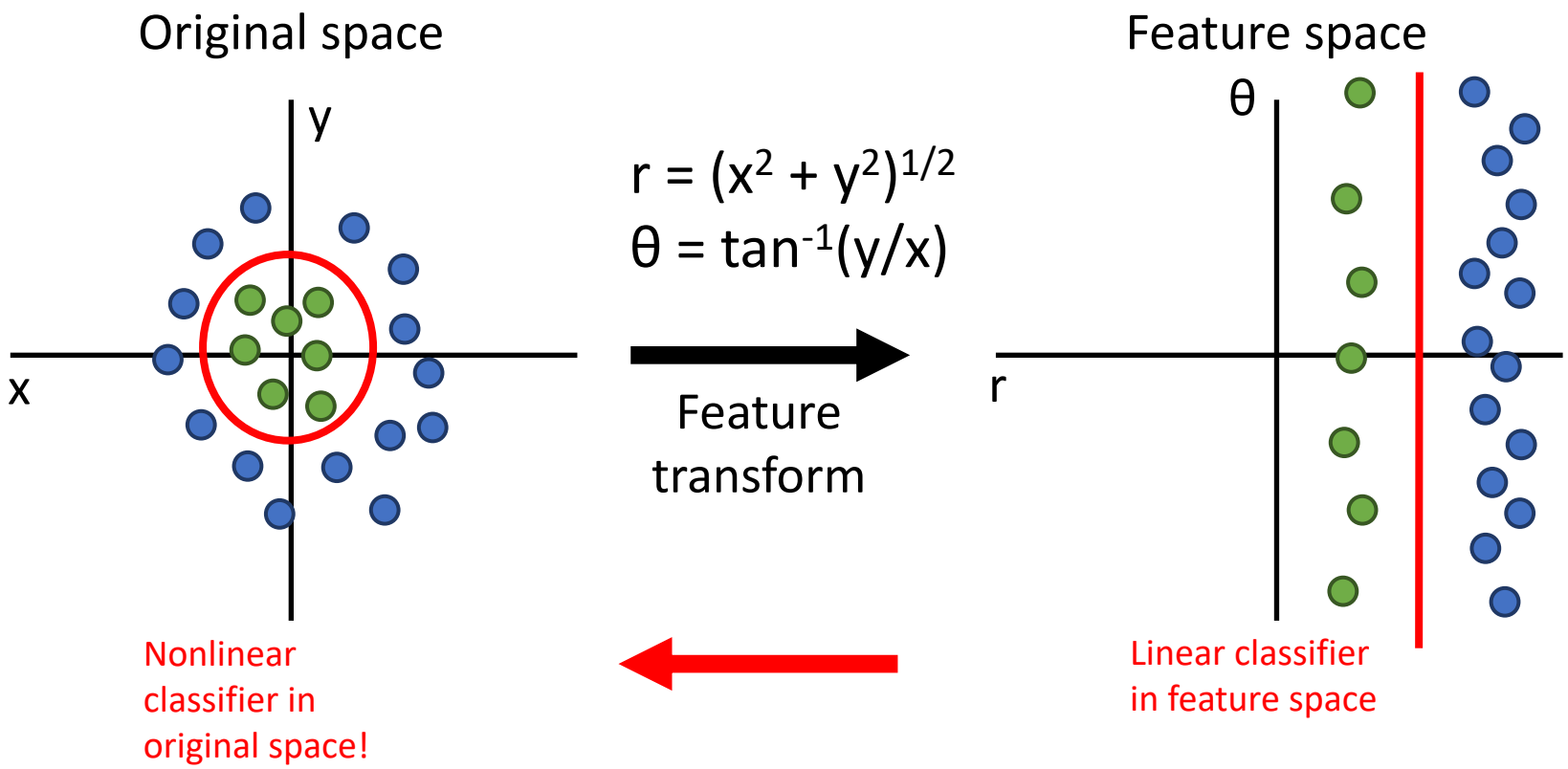
Slide credit: D Fouhey & J Johnson

One solution: **Feature Transforms**



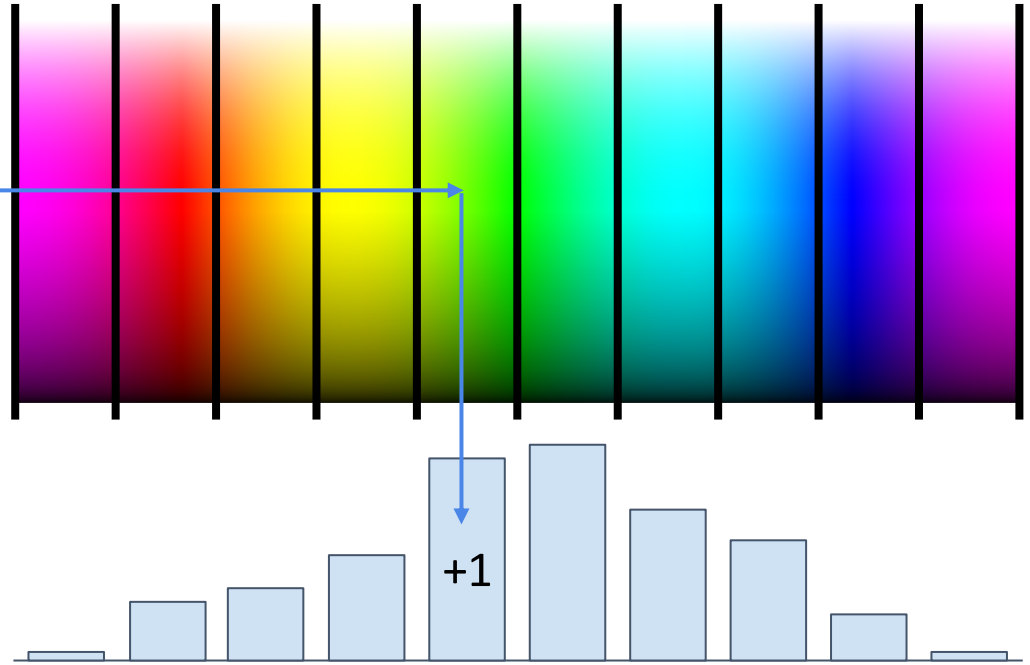
Slide credit: D Fouhey & J Johnson

One solution: **Feature Transforms**



Slide credit: D Fouhey & J Johnson

Image Features: Color Histogram



Ignores texture,
spatial positions

Slide credit: D Fouhey & J Johnson

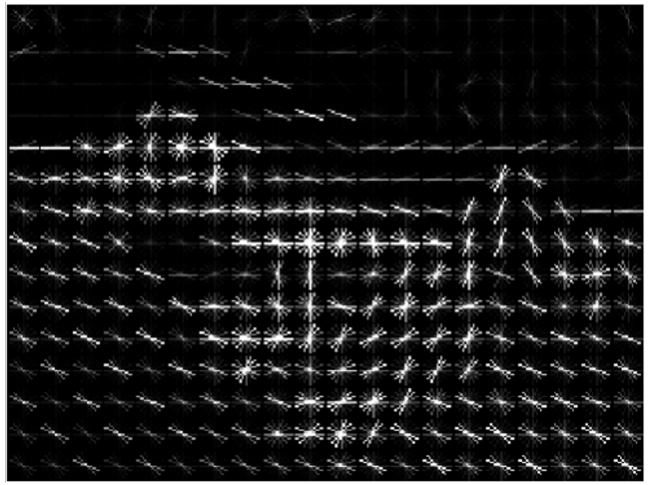
Image Features: Histogram of Oriented Gradients (HoG)

1. Compute edge direction / strength at each pixel
2. Divide image into 8x8 regions
3. Within each region compute a histogram of edge directions weighted by edge strength

Lowe, “Object recognition from local scale-invariant features”, ICCV 1999
Dalal and Triggs, "Histograms of oriented gradients for human detection," CVPR 2005

Slide credit: D Fouhey & J Johnson

Image Features: Histogram of Oriented Gradients (HoG)



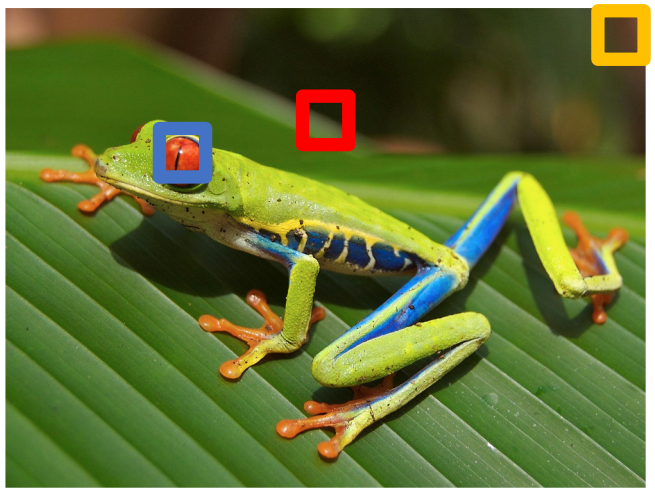
1. Compute edge direction / strength at each pixel
2. Divide image into 8x8 regions
3. Within each region compute a histogram of edge directions weighted by edge strength

Example: 320x240 image gets divided into 40x30 bins; 8 directions per bin; feature vector has 30*40*9 = 10,800 numbers

Lowe, “Object recognition from local scale-invariant features”, ICCV 1999
Dalal and Triggs, "Histograms of oriented gradients for human detection," CVPR 2005

Slide credit: D Fouhey & J Johnson

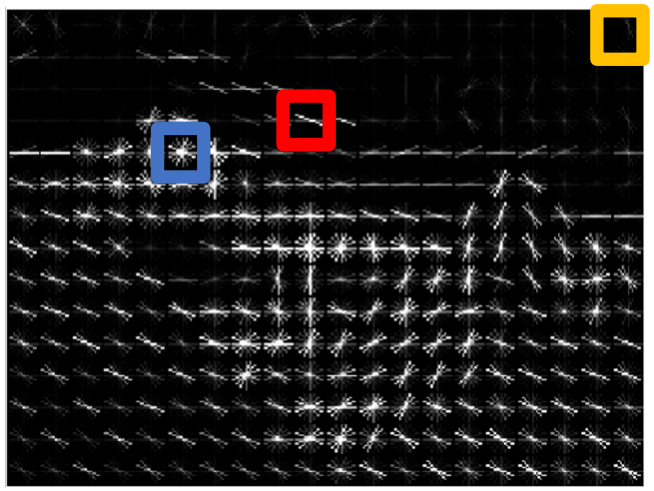
Image Features: Histogram of Oriented Gradients (HoG)



Weak edges

Strong diagonal edges

Edges in all directions



1. Compute edge direction / strength at each pixel
2. Divide image into 8x8 regions
3. Within each region compute a histogram of edge directions weighted by edge strength

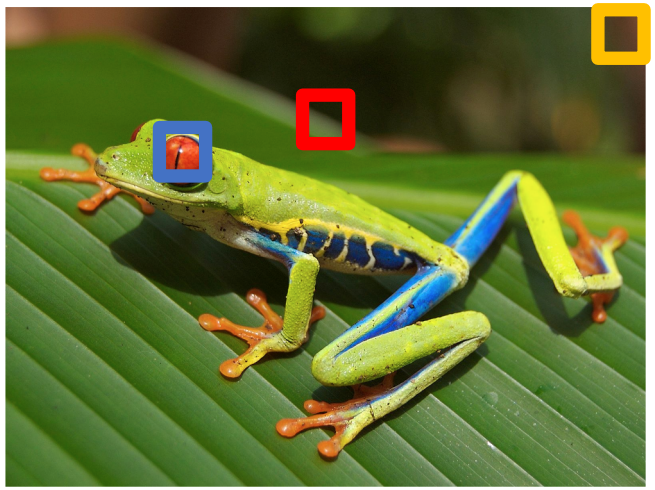
Example: 320x240 image gets divided into 40x30 bins; 8 directions per bin; feature vector has 30*40*9 = 10,800 numbers

Lowe, “Object recognition from local scale-invariant features”, ICCV 1999
Dalal and Triggs, "Histograms of oriented gradients for human detection," CVPR 2005

Slide credit: D Fouhey & J Johnson

Image Features: Histogram of Oriented Gradients (HoG)

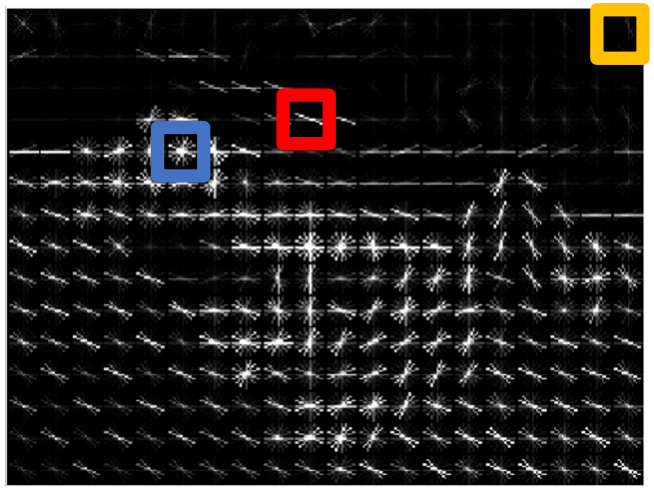
Captures texture and position, robust to small image changes



Weak edges

Strong diagonal edges

Edges in all directions



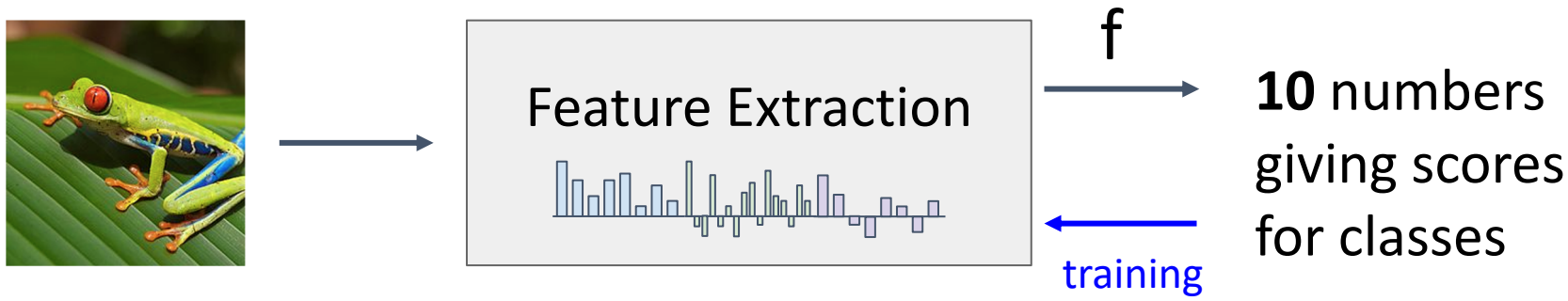
1. Compute edge direction / strength at each pixel
2. Divide image into 8x8 regions
3. Within each region compute a histogram of edge directions weighted by edge strength

Example: 320x240 image gets divided into 40x30 bins; 8 directions per bin; feature vector has 30*40*9 = 10,800 numbers

Lowe, “Object recognition from local scale-invariant features”, ICCV 1999
Dalal and Triggs, "Histograms of oriented gradients for human detection," CVPR 2005

Slide credit: D Fouhey & J Johnson

Image Features vs Neural Networks

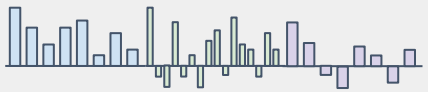


Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012

Slide credit: D Fouhey & J Johnson

Image Features vs Neural Networks

Feature Extraction

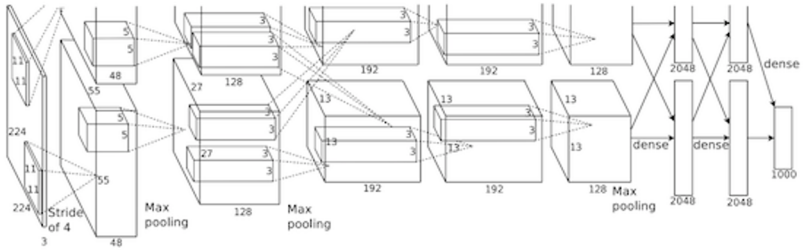


f

10 numbers giving scores for classes

training

Deep Neural Network



10 numbers giving scores for classes

training

Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012

Slide credit: D Fouhey & J Johnson

Neural Networks

Input image: $x \in \mathbb{R}^D$

Category scores: $s \in \mathbb{R}^C$

Linear Classifier:

$$s = Wx$$

$$W \in \mathbb{R}^{C \times D}$$

In practice we add a learnable bias
+b after each matrix multiply

Slide credit: D Fouhey & J Johnson

Neural Networks

Input image: $x \in \mathbb{R}^D$

Category scores: $s \in \mathbb{R}^C$

Linear Classifier:

$$s = Wx$$

$$W \in \mathbb{R}^{C \times D}$$

2-layer Neural Net:

$$s = \mathrm{W}_2 \max(0, W_1 x)$$

$$W_1 \in \mathbb{R}^{H \times D}$$

$$W_2 \in \mathbb{R}^{C \times H}$$

In practice we add a learnable bias
+b after each matrix multiply

Slide credit: D Fouhey & J Johnson

Neural Networks

Input image: $x \in \mathbb{R}^D$

Category scores: $s \in \mathbb{R}^C$

Linear Classifier:

$$s = Wx$$

$$W \in \mathbb{R}^{C \times D}$$

2-layer Neural Net:

$$s = \mathrm{W}_2 \max(0, W_1 x)$$

$$W_1 \in \mathbb{R}^{H \times D}$$

$$W_2 \in \mathbb{R}^{C \times H}$$

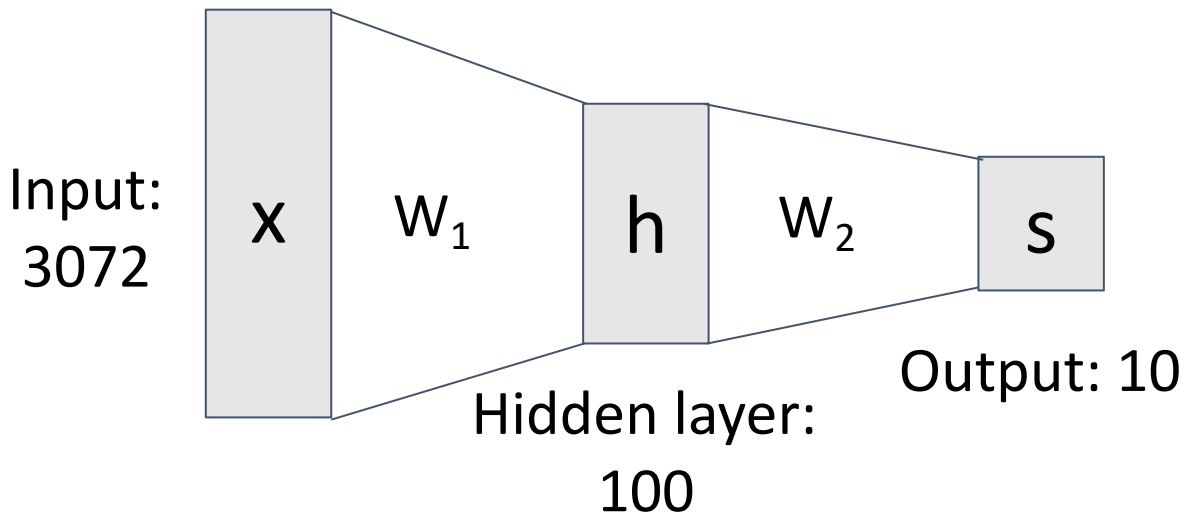
3-layer Neural Net:

$$s = \mathrm{W}_3 \max(0, W_2 \max(0, W_1 x))$$

Slide credit: D Fouhey & J Johnson

Neural Networks

Two-Layer Neural Network: $s = \mathrm{W}_2 \max(0, W_1 x)$



$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

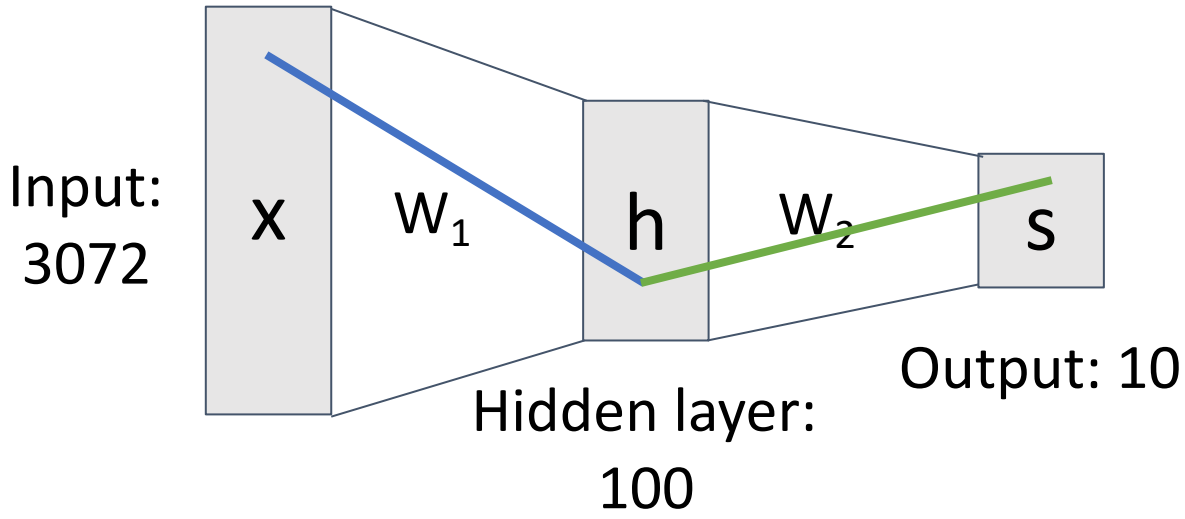
Slide credit: D Fouhey & J Johnson

Neural Networks

Two-Layer Neural Network: $s = W_2 \max(0, W_1 x)$

Element (i, j) of W_1 gives the effect on h_i from x_j

Element (i, j) of W_2 gives the effect on s_i from h_j



$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

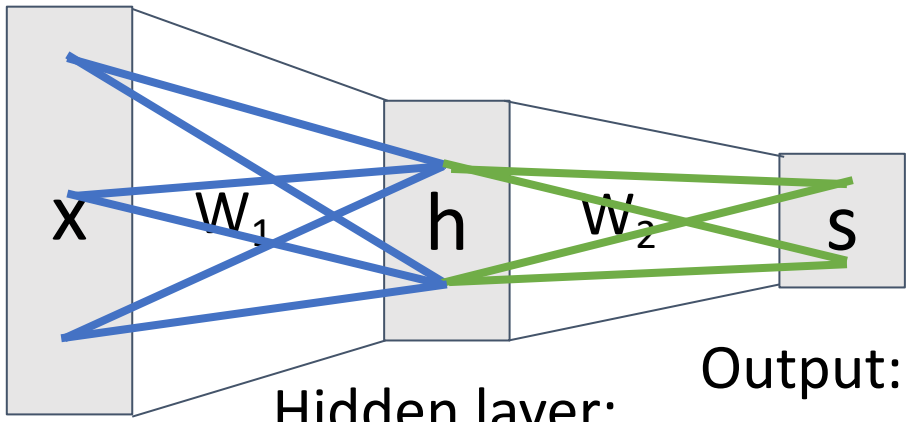
Slide credit: D Fouhey & J Johnson

Neural Networks

Two-Layer Neural Network: $s = W_2 \max(0, W_1 x)$

Element (i, j) of W_1 gives the effect on h_i from x_j

Element (i, j) of W_2 gives the effect on s_i from h_j



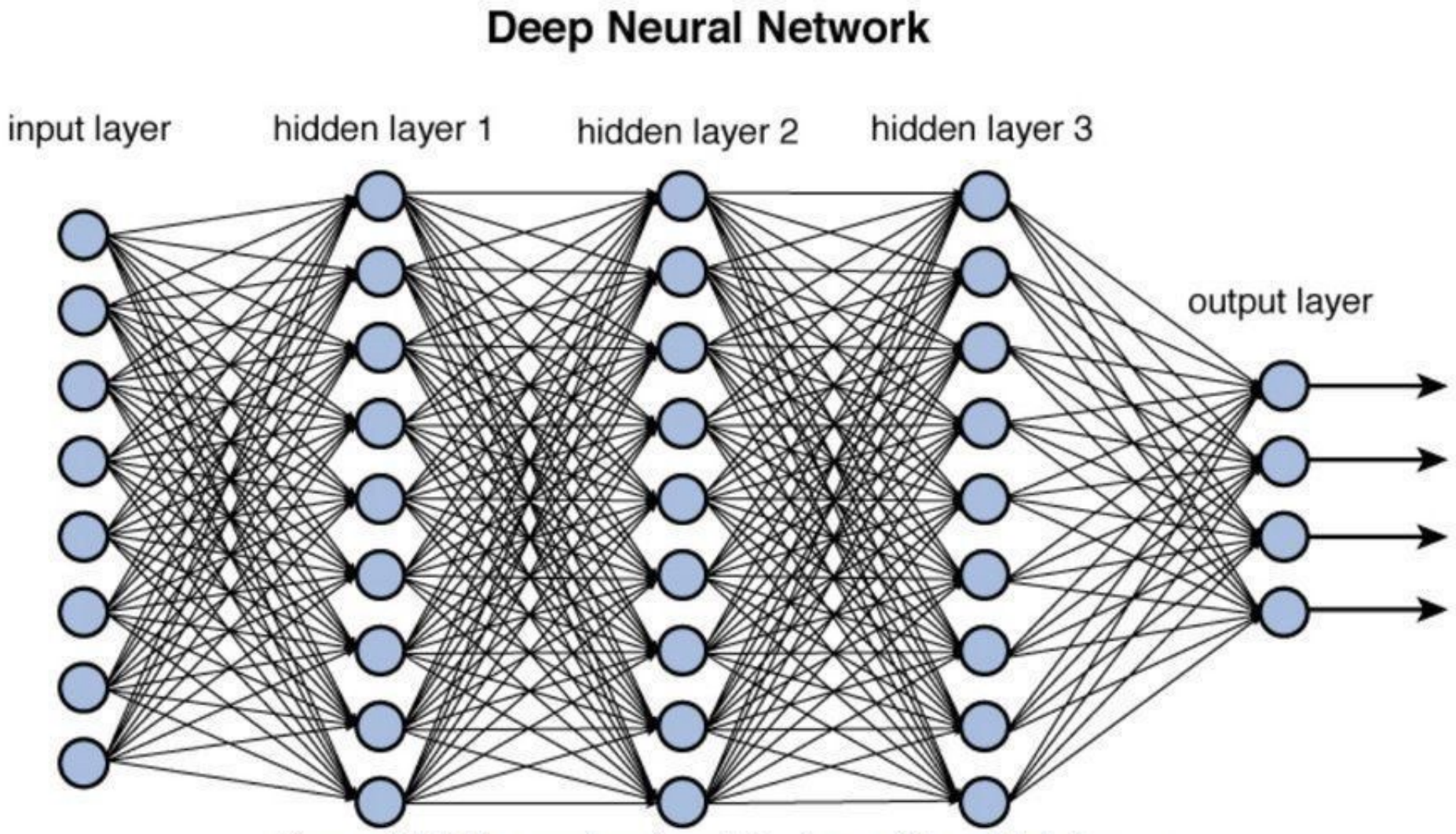
All elements of x affect all elements of h

All elements of h affect all elements of s

"Fully-Connected" neural network
Also "Multi-Layer Perceptron" (MLP)

Slide credit: D Fouhey & J Johnson

Neural Networks



Slide credit: D Fouhey & J Johnson

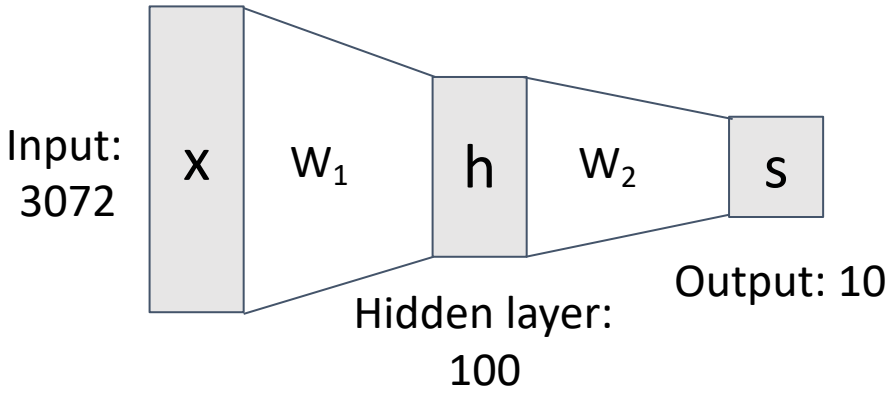
Neural Networks

Linear classifier: $s = Wx$

One template per class

Two-Layer Neural Network:

$s = \mathrm{W}_2 \max(0, W_1 x)$



$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$

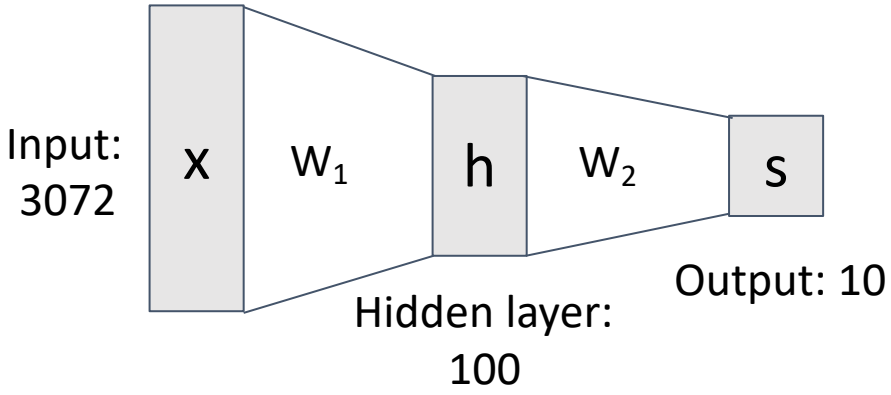
Slide credit: D Fouhey & J Johnson

Neural Networks

Neural Network:
First layer is a bank of templates
Second layer recombines templates

Two-Layer Neural Network:

$$s = W_2 \max(0, W_1 x)$$



$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

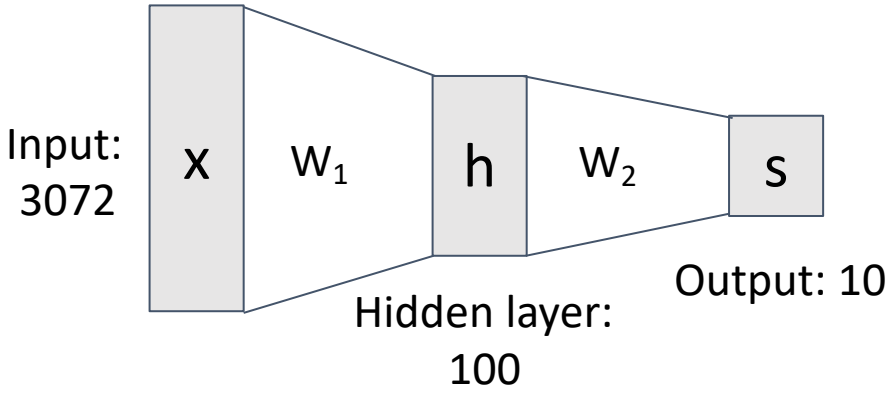
Slide credit: D Fouhey & J Johnson

Neural Networks

Different templates can cover different modes of a class!

Two-Layer Neural Network:

$$s = W_2 \max(0, W_1 x)$$



$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

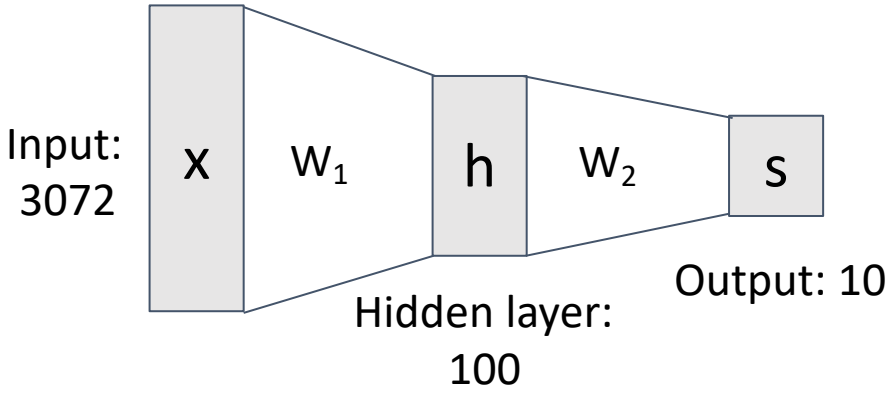
Slide credit: D Fouhey & J Johnson

Neural Networks

Many templates not interpretable:
"Distributed representation"

Two-Layer Neural Network:

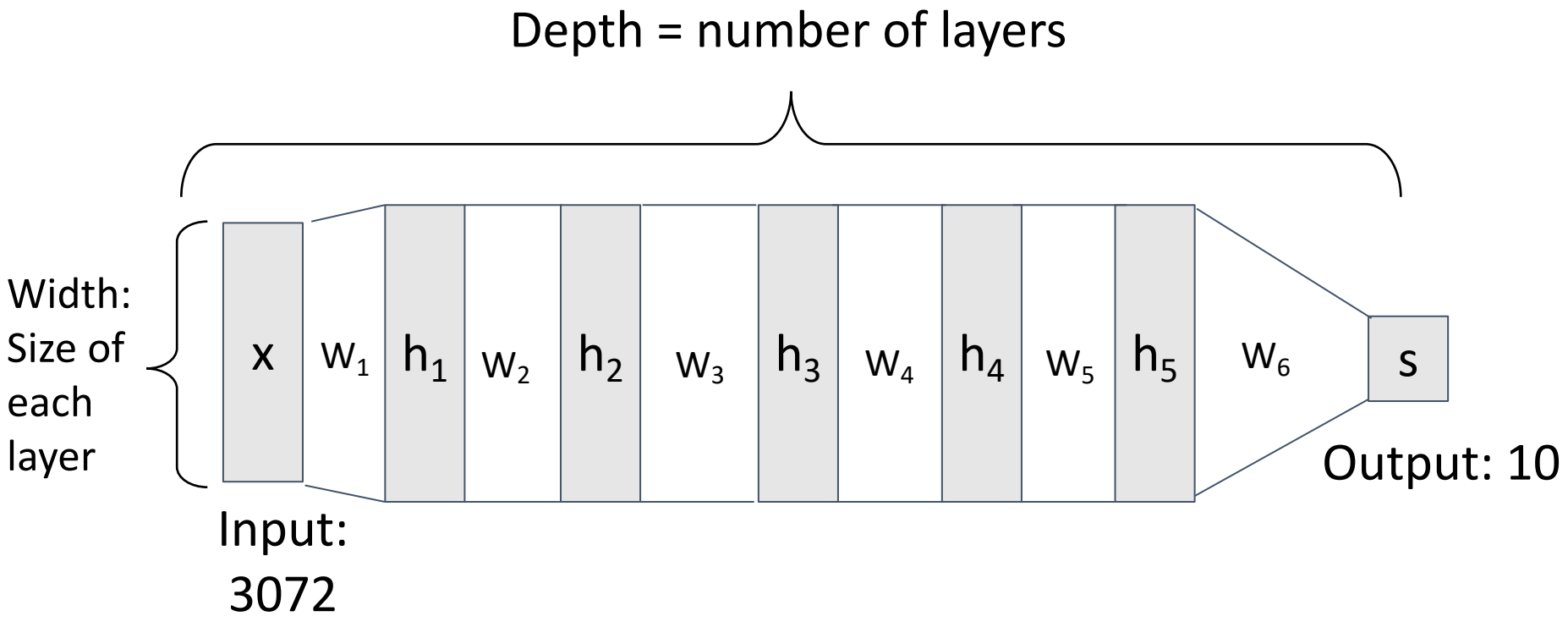
$$s = W_2 \max(0, W_1 x)$$



$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

Slide credit: D Fouhey & J Johnson

Deep Neural Networks



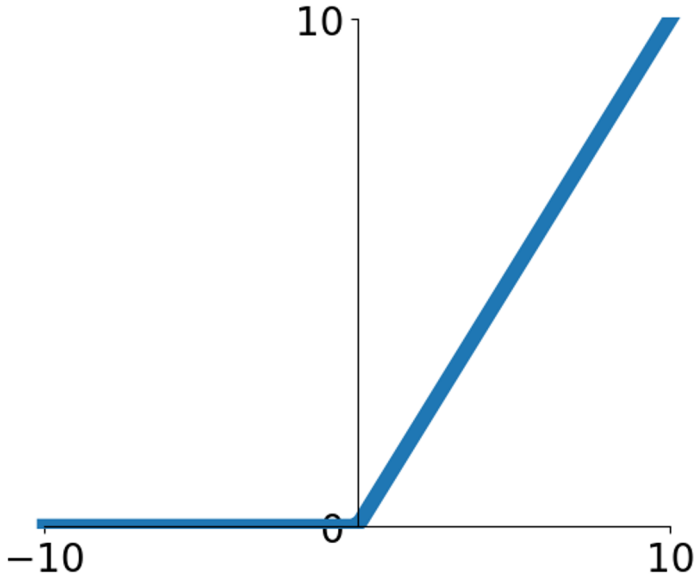
$$s = W_6 \max(0, W_5 \max(0, W_4 \max(0, W_3 \max(0, W_2 \max(0, W_1 x)))))$$

Slide credit: D Fouhey & J Johnson

Activation Functions

2-layer Neural Network

The function $ReLU(z) = \max(0, z)$ is called "Rectified Linear Unit"



$$s = W_2 \max(0, W_1 x)$$

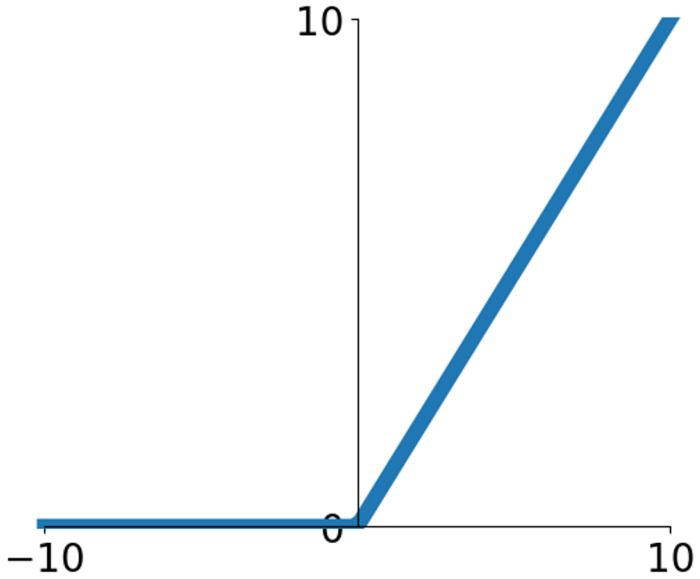
This is called the **activation function** of the neural network

Slide credit: D Fouhey & J Johnson

Activation Functions

2-layer Neural Network

The function $ReLU(z) = \max(0, z)$ is called "Rectified Linear Unit"



$$s = W_2 \max(0, W_1 x)$$

This is called the **activation function** of the neural network

Q: What happens if we build a neural network with no activation function?

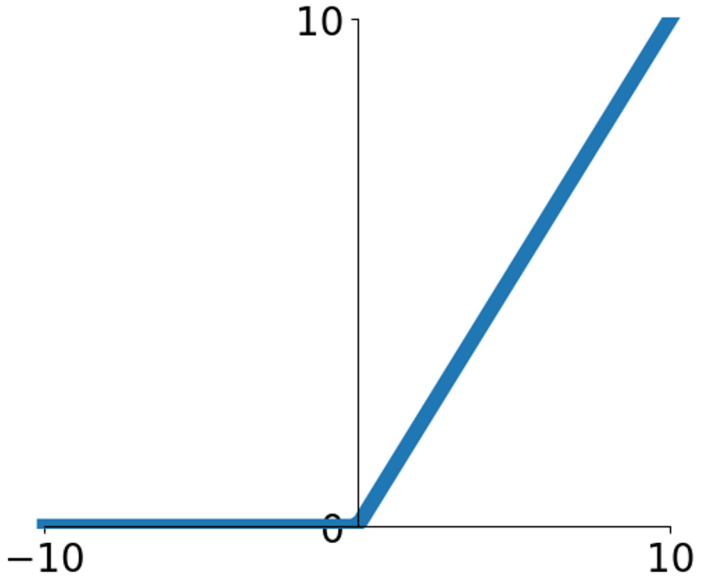
$$s = W_2 W_1 x$$

Slide credit: D Fouhey & J Johnson

Activation Functions

2-layer Neural Network

The function $ReLU(z) = \max(0, z)$ is called "Rectified Linear Unit"



$$s = W_2 \max(0, W_1 x)$$

This is called the **activation function** of the neural network

Q: What happens if we build a neural network with no activation function?

$$s = W_2 W_1 x$$

A: We get a linear classifier!

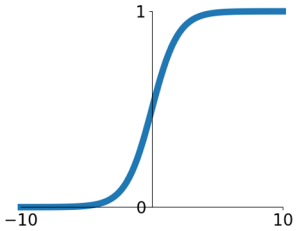
$$W_3 = W_2 W_1 \in \mathbb{R}^{C \times D}$$
$$s = W_3 x$$

Slide credit: D Fouhey & J Johnson

Activation Functions

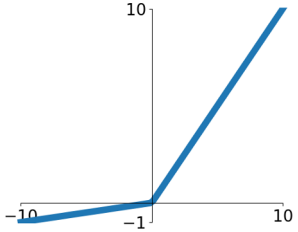
Sigmoid

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



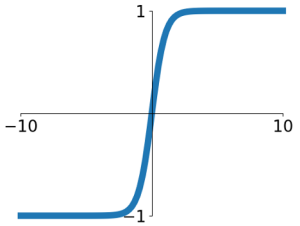
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

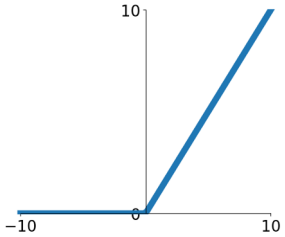


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

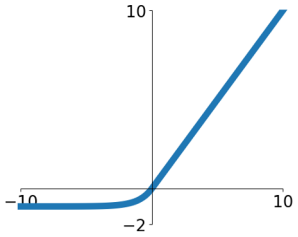
ReLU

$$\max(0, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



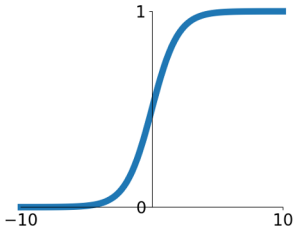
Slide credit: D Fouhey & J Johnson

Activation Functions

ReLU is a good default choice

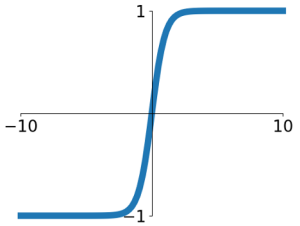
Sigmoid

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



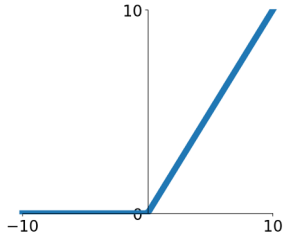
tanh

$$\tanh(x)$$



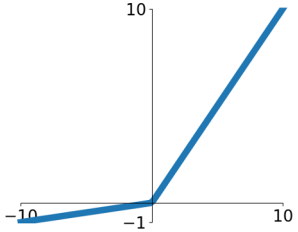
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

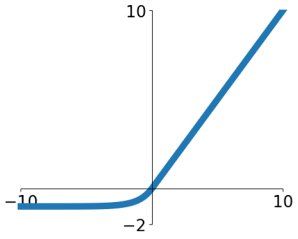


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Slide credit: D Fouhey & J Johnson

Neural Net in <20 lines!

```
1   import numpy as np
2   from numpy.random import randn
3
4   N, Din, H, Dout = 64, 1000, 100, 10
5   x, y = randn(N, Din), randn(N, Dout)
6   w1, w2 = randn(Din, H), randn(H, Dout)
7   for t in range(10000):
8     h = 1.0 / (1.0 + np.exp(-x.dot(w1)))
9     y_pred = h.dot(w2)
10    loss = np.square(y_pred - y).sum()
11    dy_pred = 2.0 * (y_pred - y)
12    dw2 = h.T.dot(dy_pred)
13    dh = dy_pred.dot(w2.T)
14    dw1 = x.T.dot(dh * h * (1 - h))
15    w1 -= 1e-4 * dw1
16    w2 -= 1e-4 * dw2
```

Slide credit: D Fouhey & J Johnson

Neural Net in <20 lines!

Initialize weights and data

```
1   import numpy as np
2   from numpy.random import randn
3
4   N, Din, H, Dout = 64, 1000, 100, 10
5   x, y = randn(N, Din), randn(N, Dout)
6   w1, w2 = randn(Din, H), randn(H, Dout)
7   for t in range(10000):
8     h = 1.0 / (1.0 + np.exp(-x.dot(w1)))
9     y_pred = h.dot(w2)
10    loss = np.square(y_pred - y).sum()
11    dy_pred = 2.0 * (y_pred - y)
12    dw2 = h.T.dot(dy_pred)
13    dh = dy_pred.dot(w2.T)
14    dw1 = x.T.dot(dh * h * (1 - h))
15    w1 -= 1e-4 * dw1
16    w2 -= 1e-4 * dw2
```

Slide credit: D Fouhey & J Johnson

Neural Net in <20 lines!

Initialize weights and data

Compute loss (sigmoid activation, L2 loss)

```
1   import numpy as np
2   from numpy.random import randn
3
4   N, Din, H, Dout = 64, 1000, 100, 10
5   x, y = randn(N, Din), randn(N, Dout)
6   w1, w2 = randn(Din, H), randn(H, Dout)
7   for t in range(10000):
8     h = 1.0 / (1.0 + np.exp(-x.dot(w1)))
9     y_pred = h.dot(w2)
10    loss = np.square(y_pred - y).sum()
11    dy_pred = 2.0 * (y_pred - y)
12    dw2 = h.T.dot(dy_pred)
13    dh = dy_pred.dot(w2.T)
14    dw1 = x.T.dot(dh * h * (1 - h))
15    w1 -= 1e-4 * dw1
16    w2 -= 1e-4 * dw2
```

Slide credit: D Fouhey & J Johnson

Neural Net in <20 lines!

Initialize weights and data

Compute loss (sigmoid activation, L2 loss)

Compute gradients

```
1   import numpy as np
2   from numpy.random import randn
3
4   N, Din, H, Dout = 64, 1000, 100, 10
5   x, y = randn(N, Din), randn(N, Dout)
6   w1, w2 = randn(Din, H), randn(H, Dout)
7   for t in range(10000):
8     h = 1.0 / (1.0 + np.exp(-x.dot(w1)))
9     y_pred = h.dot(w2)
10    loss = np.square(y_pred - y).sum()
11    dy_pred = 2.0 * (y_pred - y)
12    dw2 = h.T.dot(dy_pred)
13    dh = dy_pred.dot(w2.T)
14    dw1 = x.T.dot(dh * h * (1 - h))
15    w1 -= 1e-4 * dw1
16    w2 -= 1e-4 * dw2
```

Slide credit: D Fouhey & J Johnson

Neural Net in <20 lines!

Initialize weights and data

Compute loss (sigmoid activation, L2 loss)

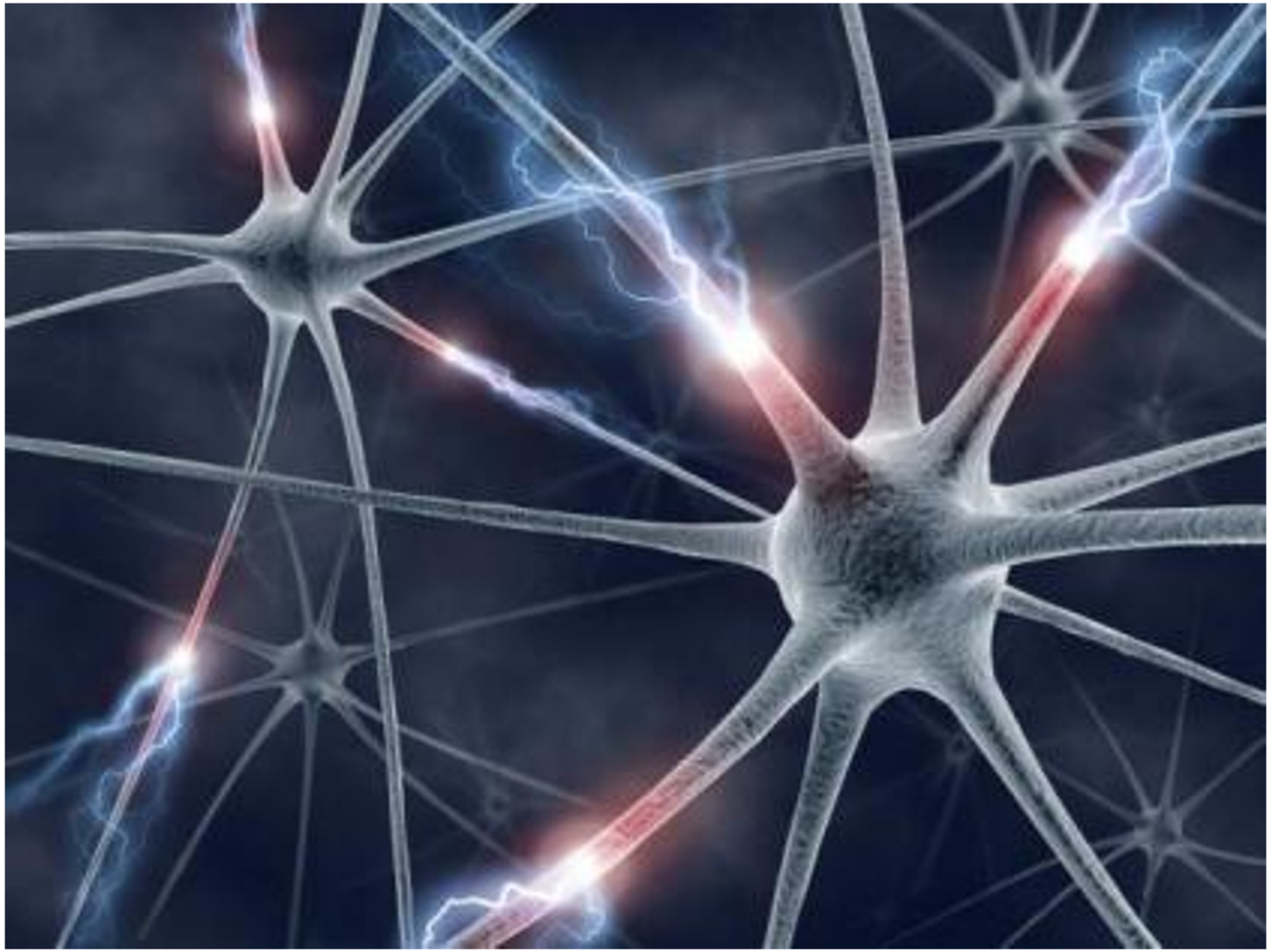
Compute gradients

SGD step

```
1   import numpy as np
2   from numpy.random import randn
3
4   N, Din, H, Dout = 64, 1000, 100, 10
5   x, y = randn(N, Din), randn(N, Dout)
6   w1, w2 = randn(Din, H), randn(H, Dout)
7   for t in range(10000):
8     h = 1.0 / (1.0 + np.exp(-x.dot(w1)))
9     y_pred = h.dot(w2)
10    loss = np.square(y_pred - y).sum()
11    dy_pred = 2.0 * (y_pred - y)
12    dw2 = h.T.dot(dy_pred)
13    dh = dy_pred.dot(w2.T)
14    dw1 = x.T.dot(dh * h * (1 - h))
15    w1 -= 1e-4 * dw1
16    w2 -= 1e-4 * dw2
```

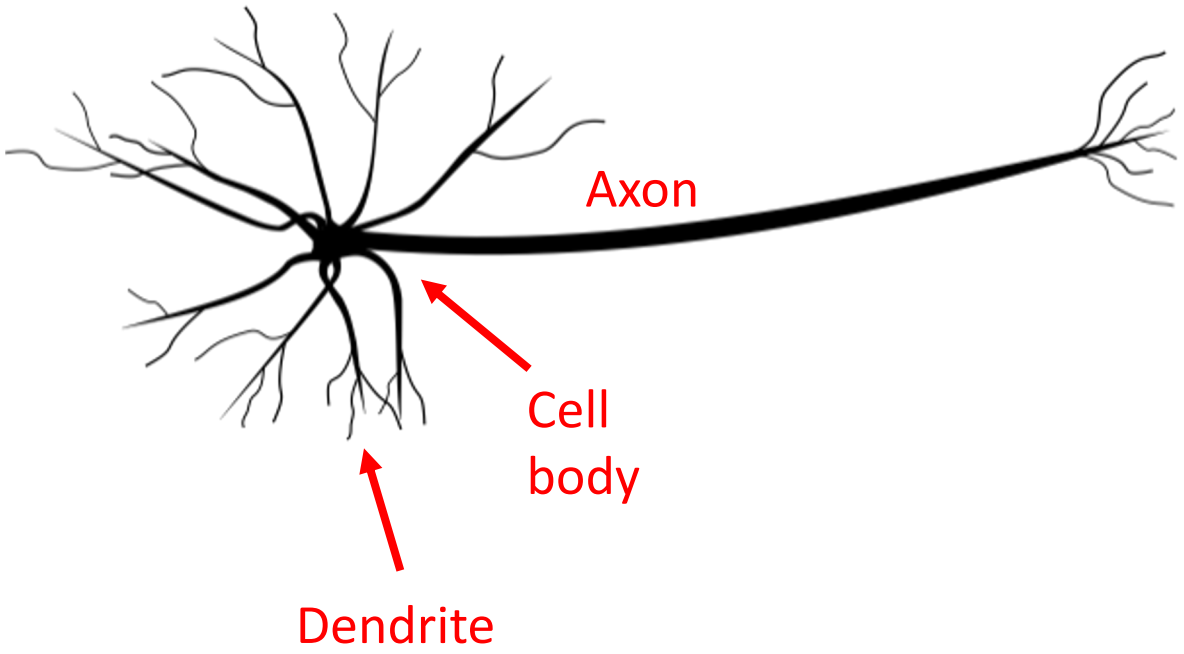
Slide credit: D Fouhey & J Johnson

“Neural” Networks



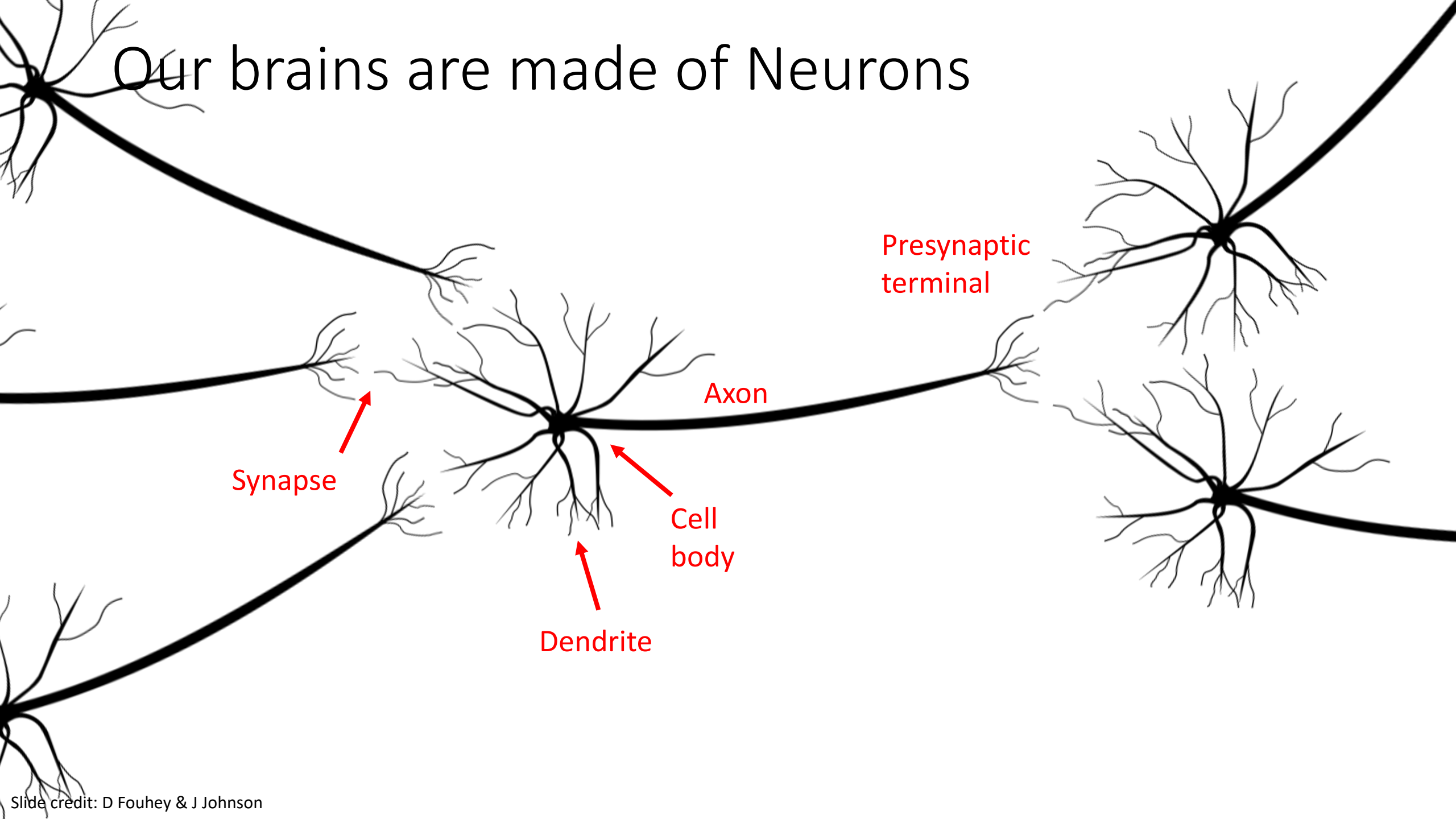
Slide credit: D Fouhey & J Johnson

Our brains are made of Neurons



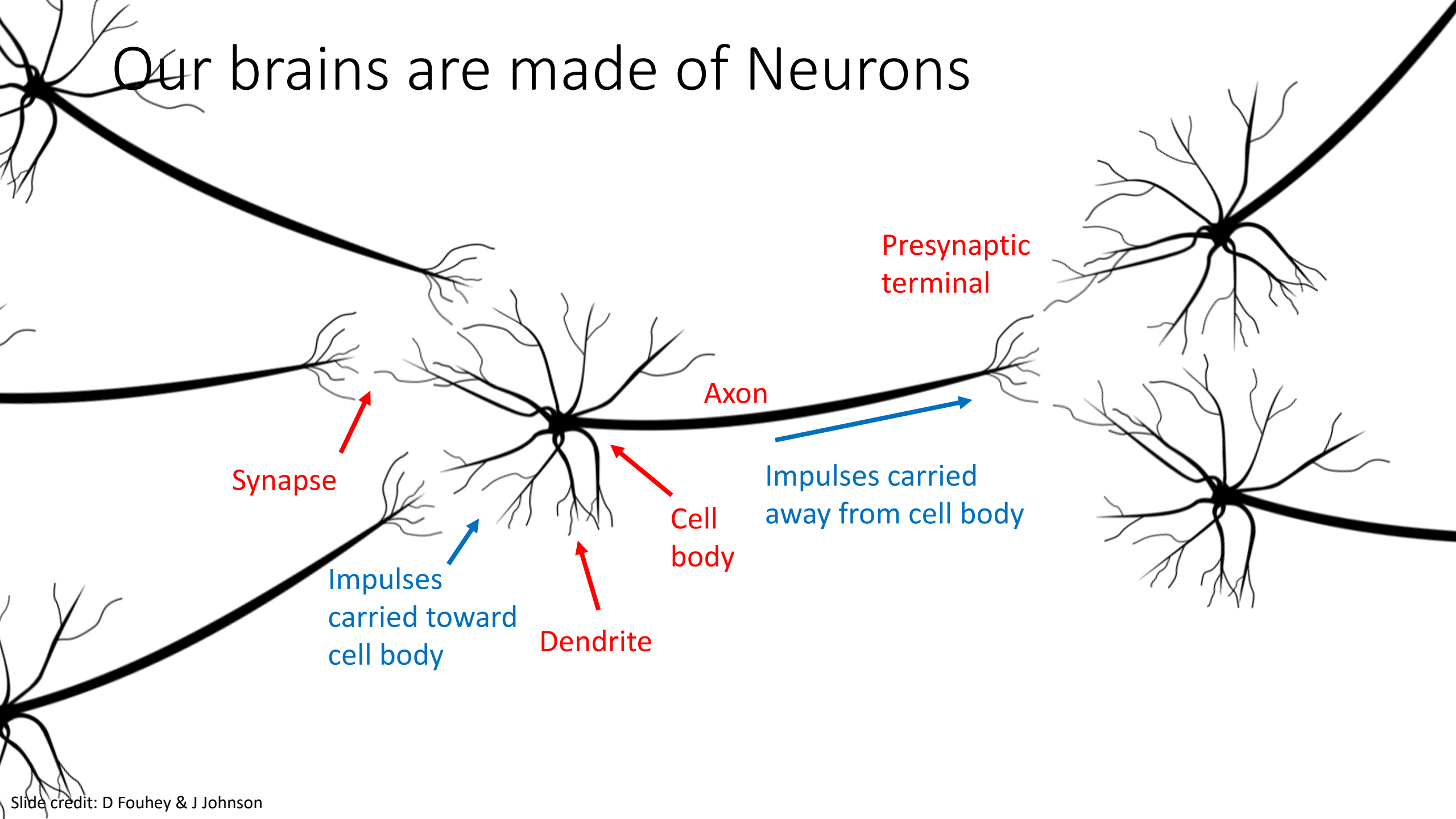
Slide credit: D Fouhey & J Johnson

Our brains are made of Neurons



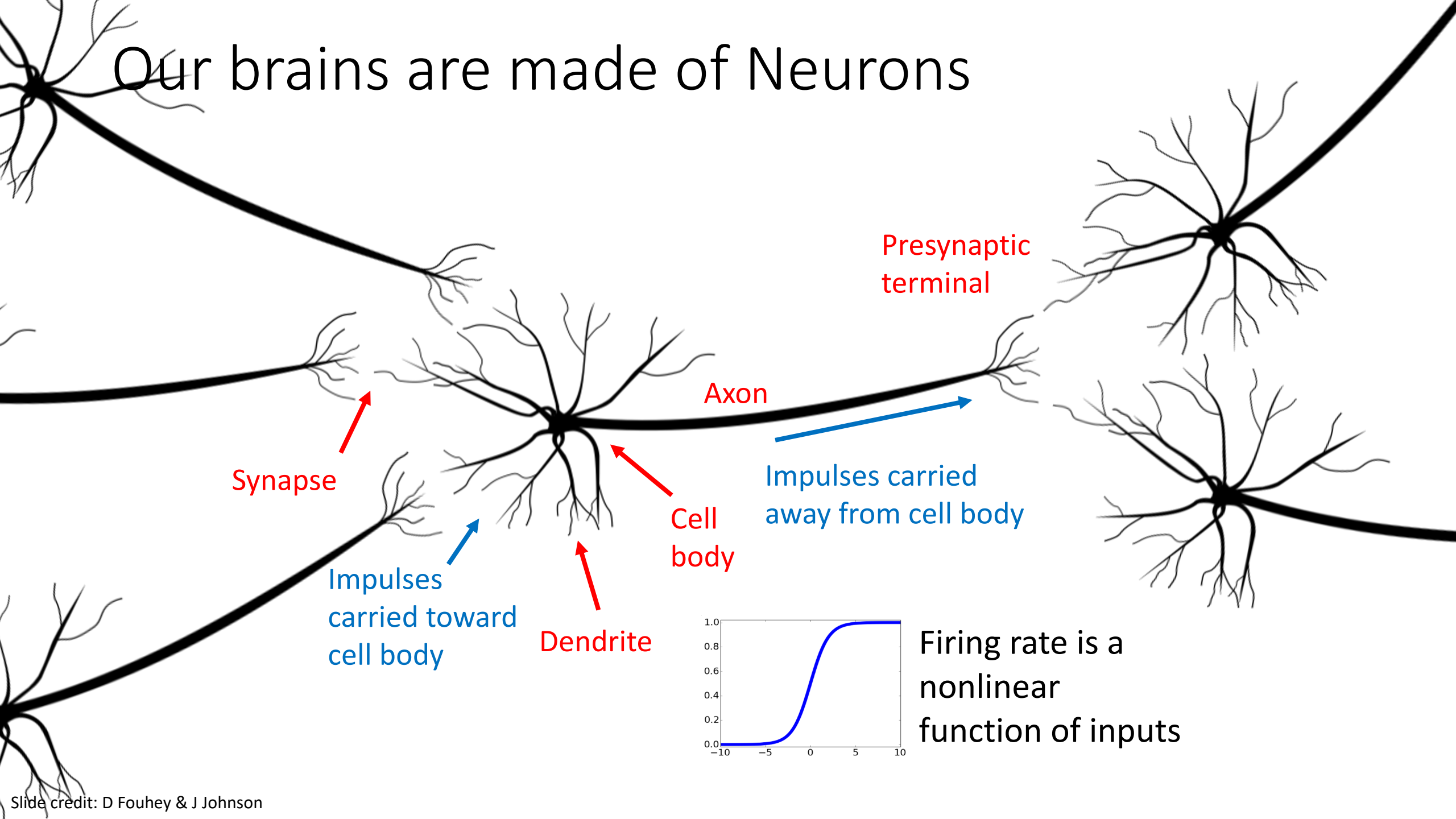
Slide credit: D Fouhey & J Johnson

Our brains are made of Neurons



Slide credit: D Fouhey & J Johnson

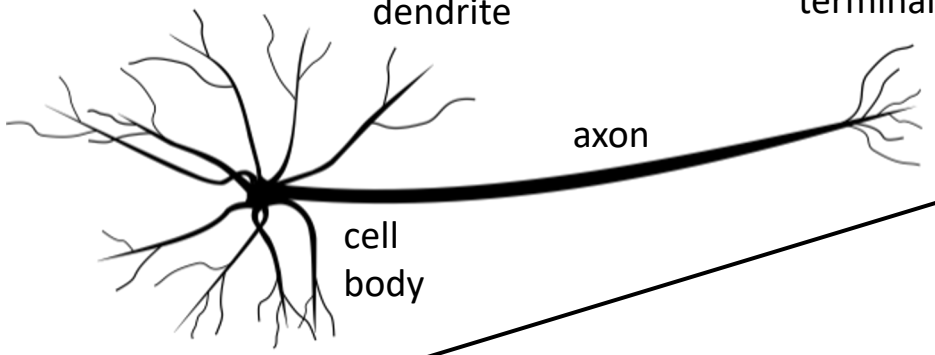
Our brains are made of Neurons



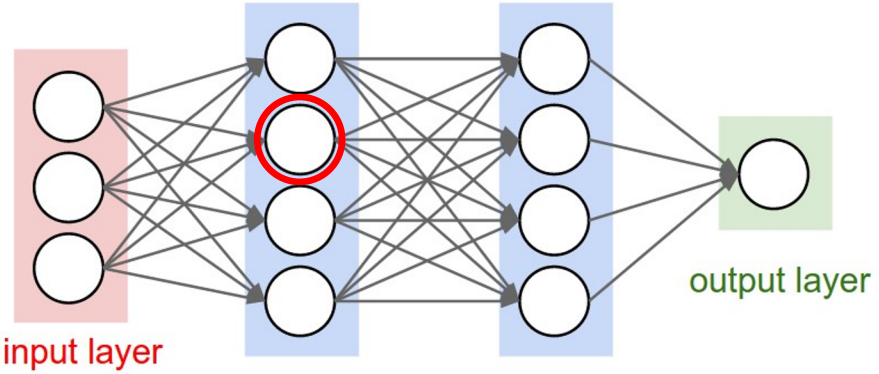
Firing rate is a nonlinear function of inputs

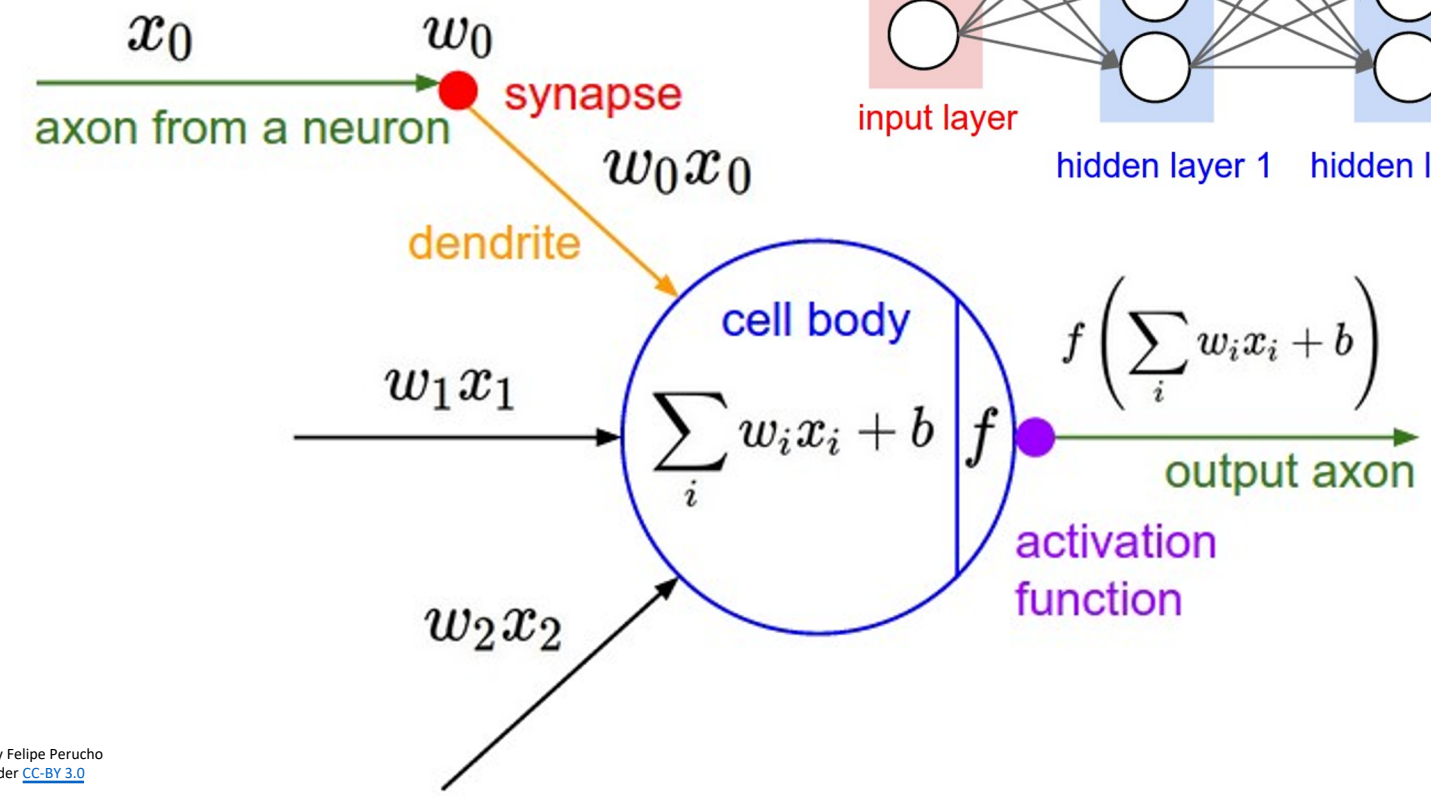
Slide credit: D Fouhey & J Johnson

Biological Neuron



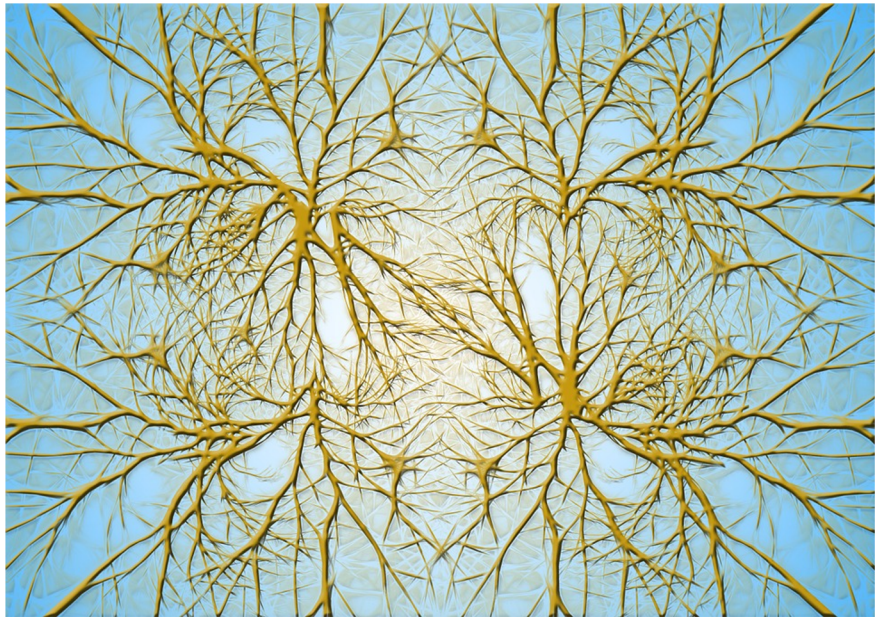
Artificial Neuron





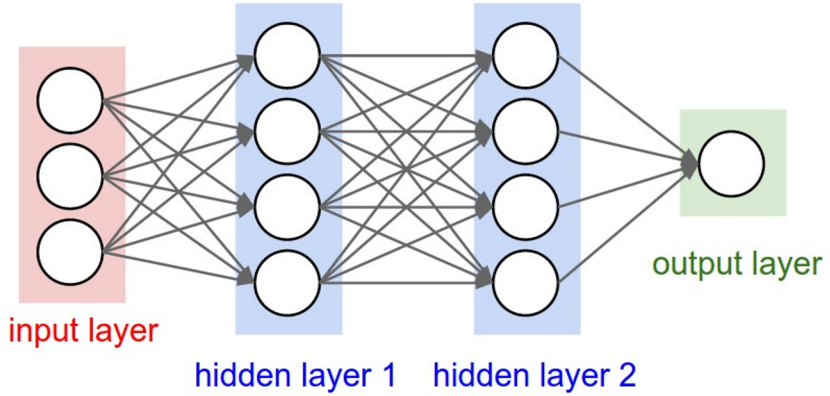
Slide credit: D Fouhey & J Johnson

Biological Neurons:
Complex connectivity patterns



This image is CC0 Public Domain

Neurons in a neural network:
Organized into regular layers
for computational efficiency



Slide credit: D Fouhey & J Johnson

Be very careful with brain analogies!

Biological Neurons:

- Many different types
- Can perform complex non-linear computations
- Synapses are not a single weight but a complex non-linear dynamical system
- Can have feedback, time-dependent
- Probably don’t learn via gradient descent

[Dendritic Computation. London and Hausser]

Slide credit: D Fouhey & J Johnson

Next Class

- Backpropagation (how to compute gradients for a neural network?)