Deep Learning (1470)

Randall Balestriero

Class 5: Optimization and Hyper-parameters

Rewind

Rewind

- Can a Deep Network solve anything?

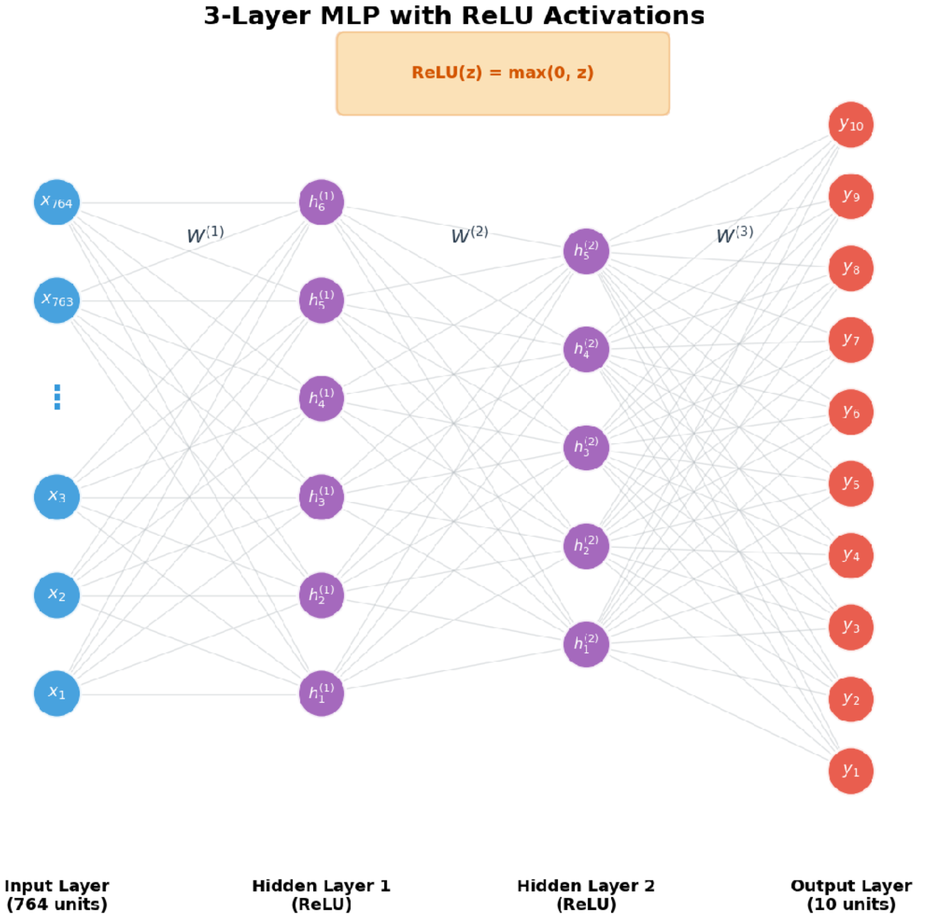
Rewind

- Can a Deep Network solve anything?
- How to search for the right “architecture”?

Rewind

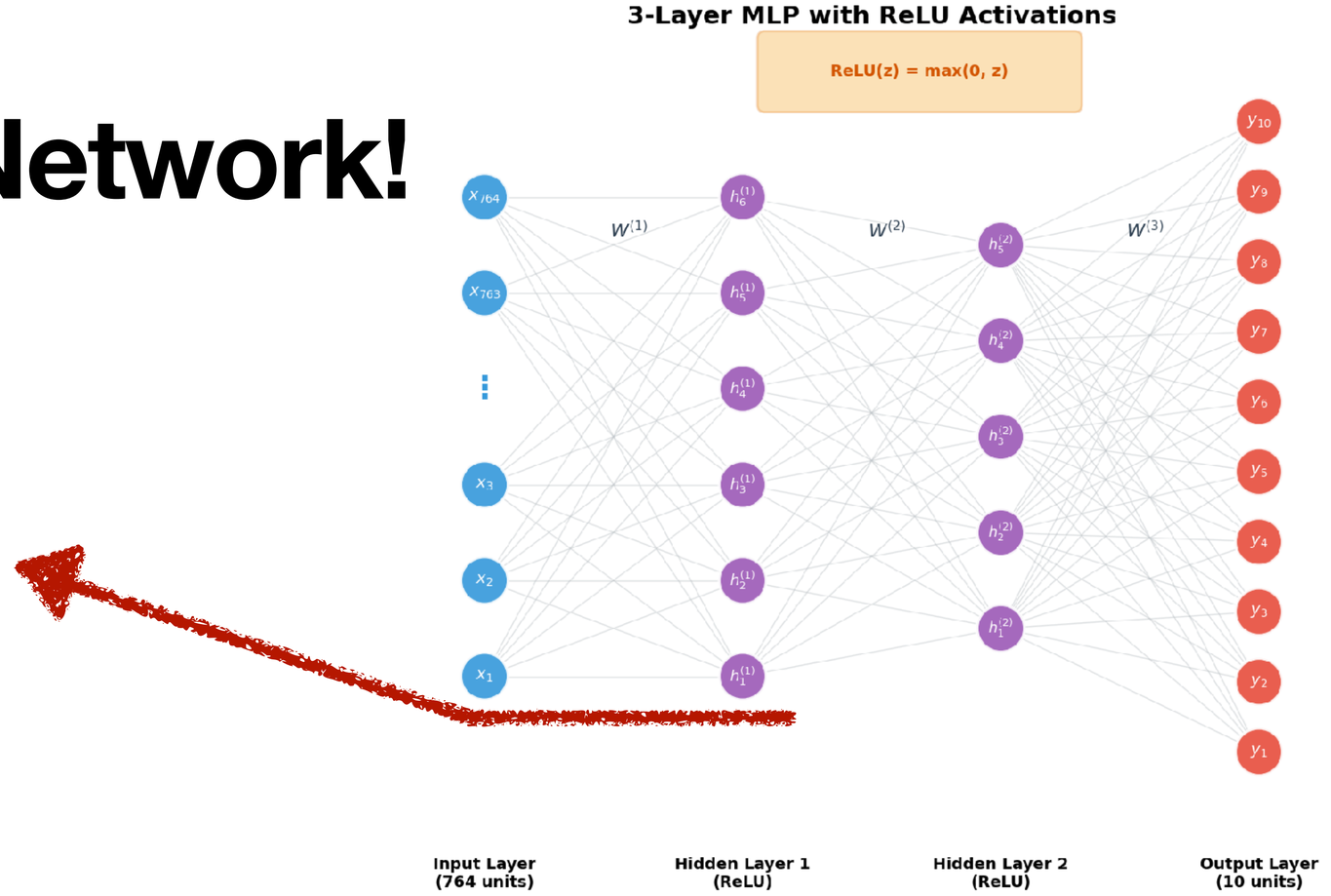
- Can a Deep Network solve anything?
- How to search for the right “architecture”?
- How many lines of codes to implement the MNIST model and reach 99.5%?

Your First Deep Network!



Your First Deep Network!

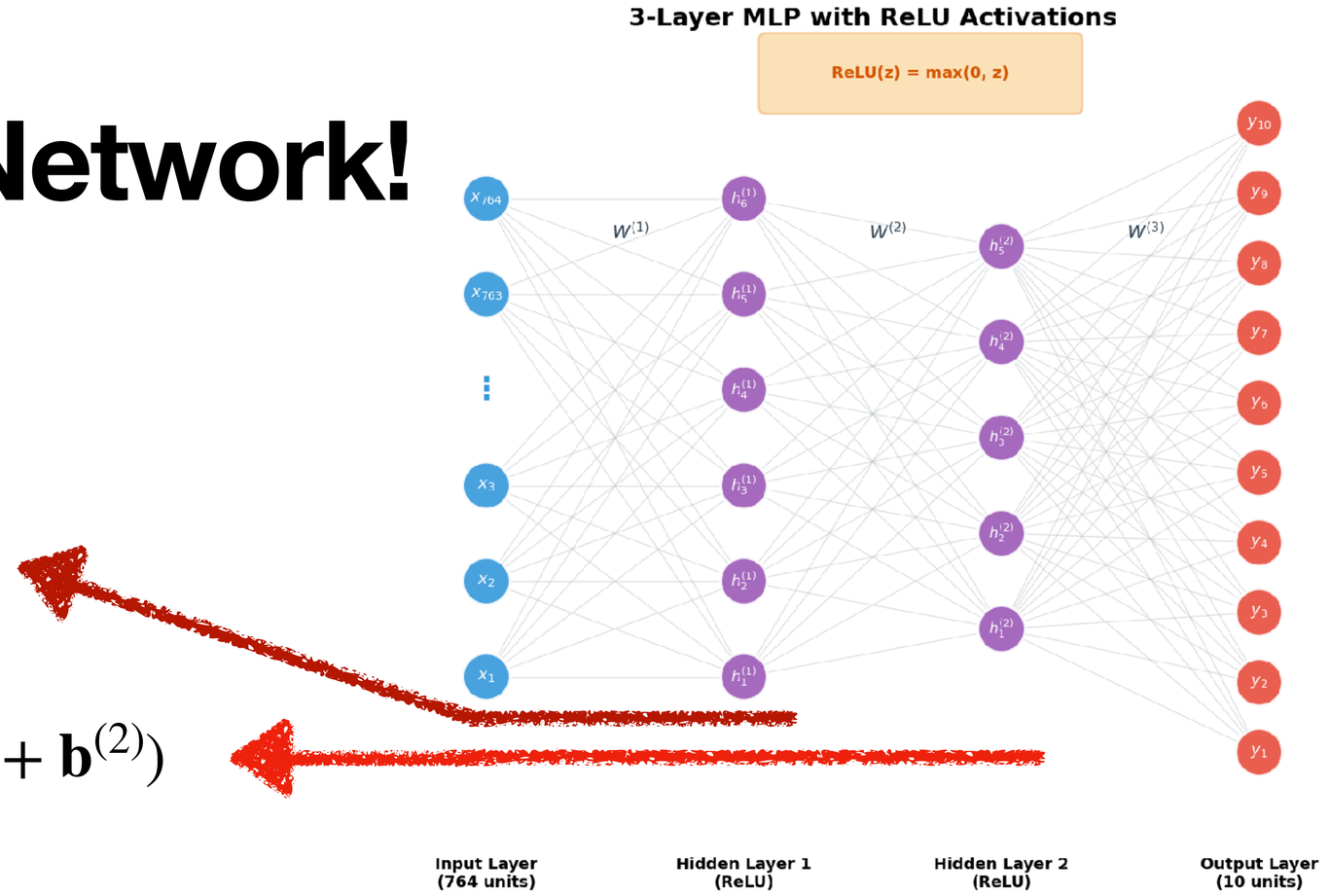
$$\text{ReLU}(\mathbf{W}^{(1)}\mathbf{x}_n + \mathbf{b}^{(1)})$$



Your First Deep Network!

$$\text{ReLU}(\mathbf{W}^{(1)}\mathbf{x}_n + \mathbf{b}^{(1)})$$

$$\text{ReLU}(\mathbf{W}^{(2)}\text{ReLU}(\mathbf{W}^{(1)}\mathbf{x}_n + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)})$$

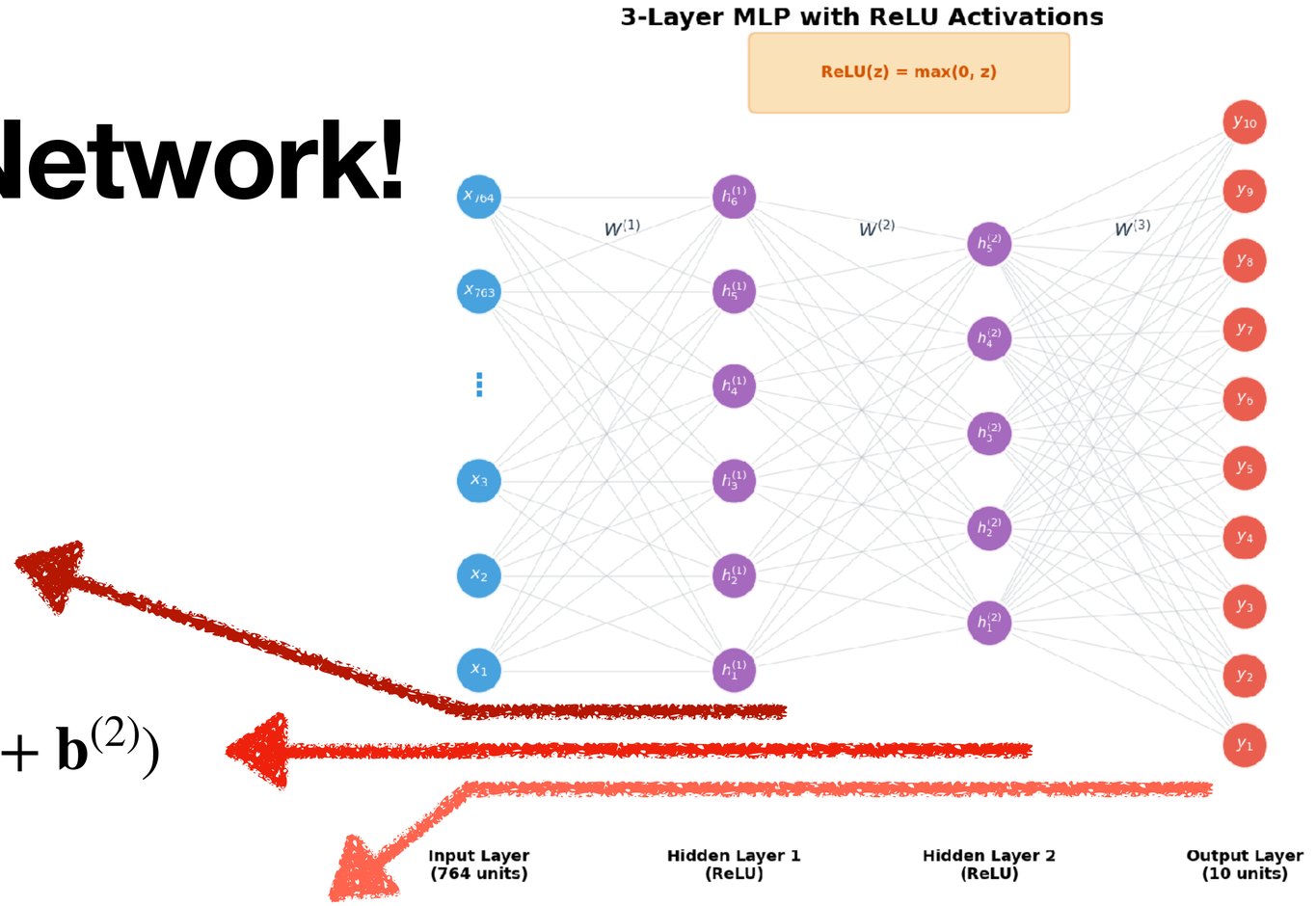


Your First Deep Network!

$$\text{ReLU}(\mathbf{W}^{(1)}\mathbf{x}_n + \mathbf{b}^{(1)})$$

$$\text{ReLU}(\mathbf{W}^{(2)}\text{ReLU}(\mathbf{W}^{(1)}\mathbf{x}_n + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)})$$

$$\mathbf{W}^{(3)}\text{ReLU}(\mathbf{W}^{(2)}\text{ReLU}(\mathbf{W}^{(1)}\mathbf{x}_n + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) + \mathbf{b}^{(3)}$$

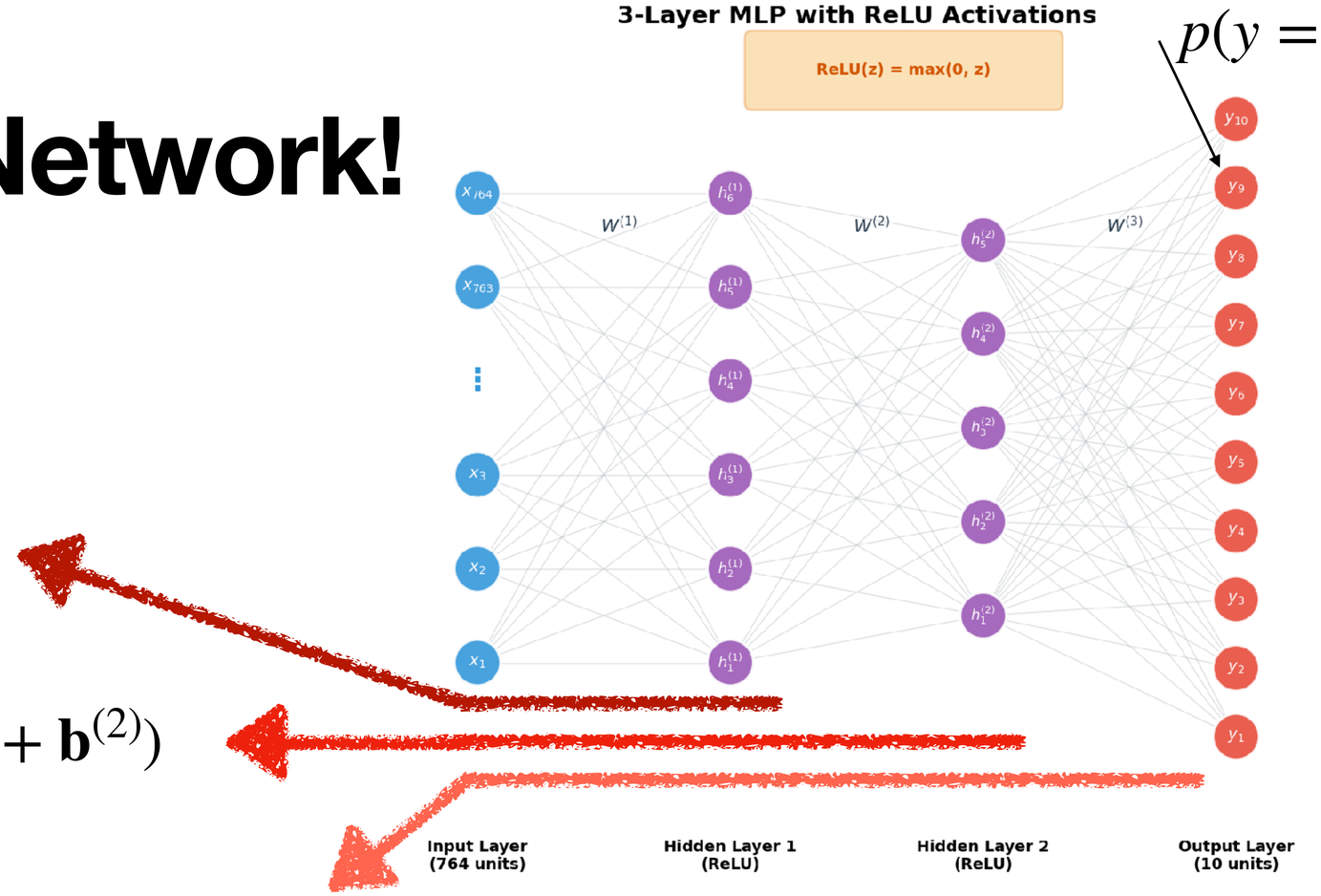


Your First Deep Network!

$$\text{ReLU}(\mathbf{W}^{(1)}\mathbf{x}_n + \mathbf{b}^{(1)})$$

$$\text{ReLU}(\mathbf{W}^{(2)}\text{ReLU}(\mathbf{W}^{(1)}\mathbf{x}_n + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)})$$

$$\mathbf{W}^{(3)}\text{ReLU}(\mathbf{W}^{(2)}\text{ReLU}(\mathbf{W}^{(1)}\mathbf{x}_n + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) + \mathbf{b}^{(3)}$$

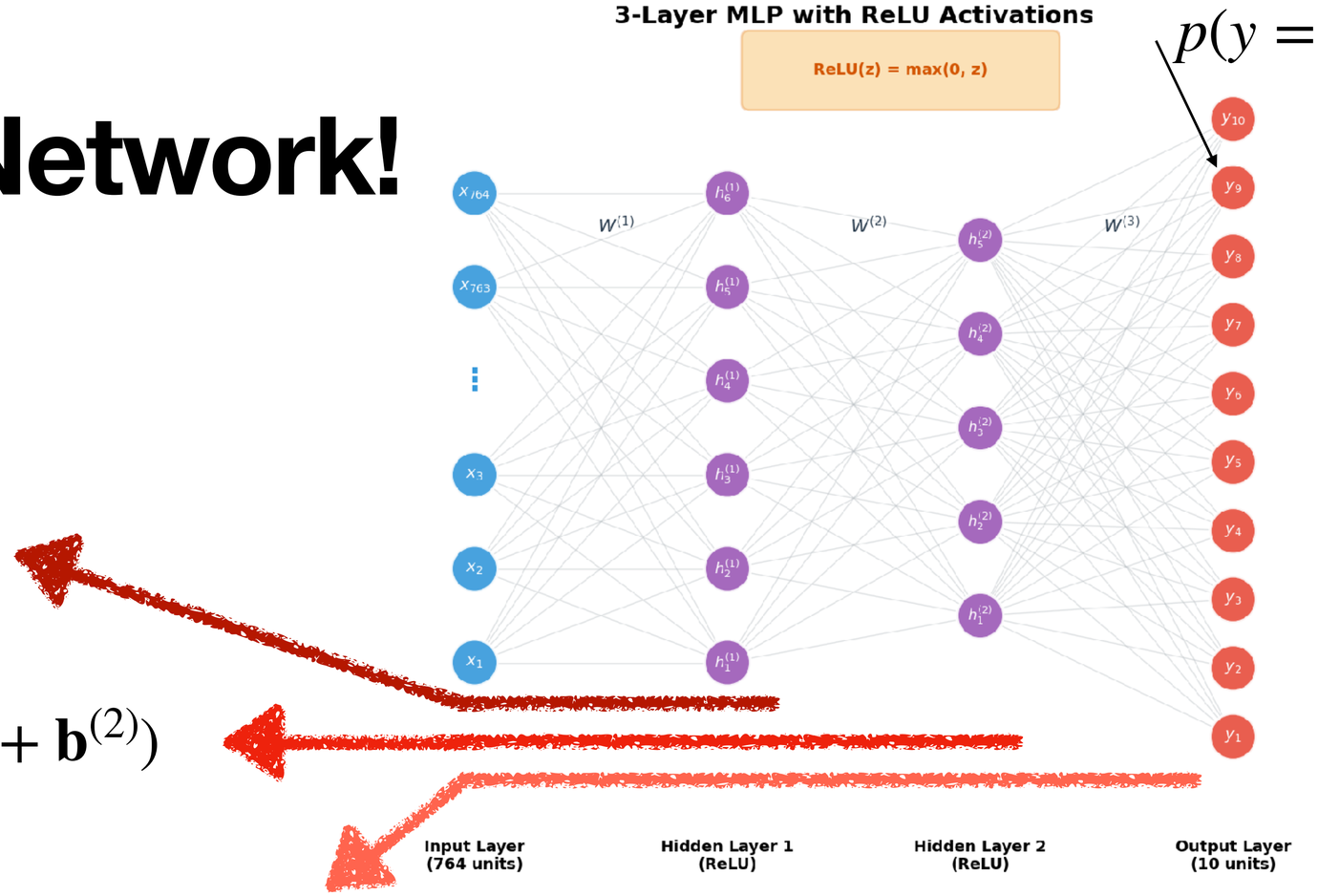


Your First Deep Network!

$$\text{ReLU}(\mathbf{W}^{(1)}\mathbf{x}_n + \mathbf{b}^{(1)})$$

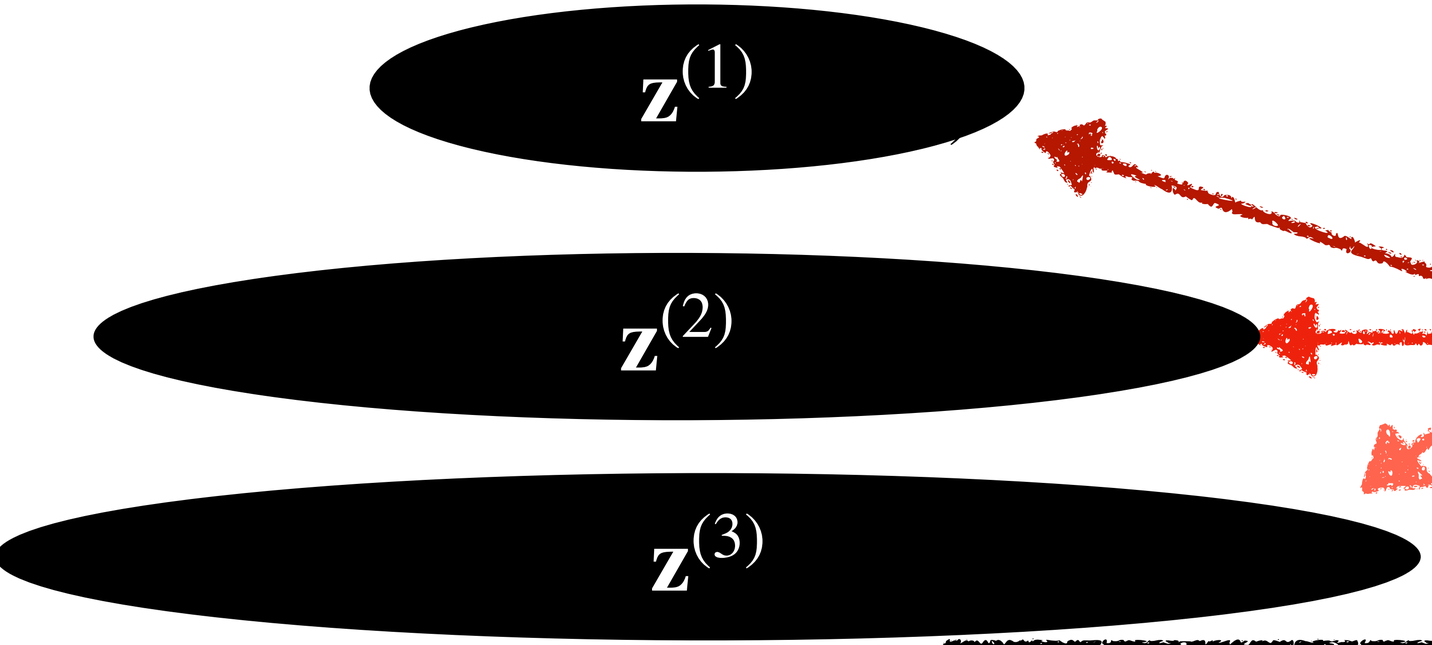
$$\text{ReLU}(\mathbf{W}^{(2)}\text{ReLU}(\mathbf{W}^{(1)}\mathbf{x}_n + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)})$$

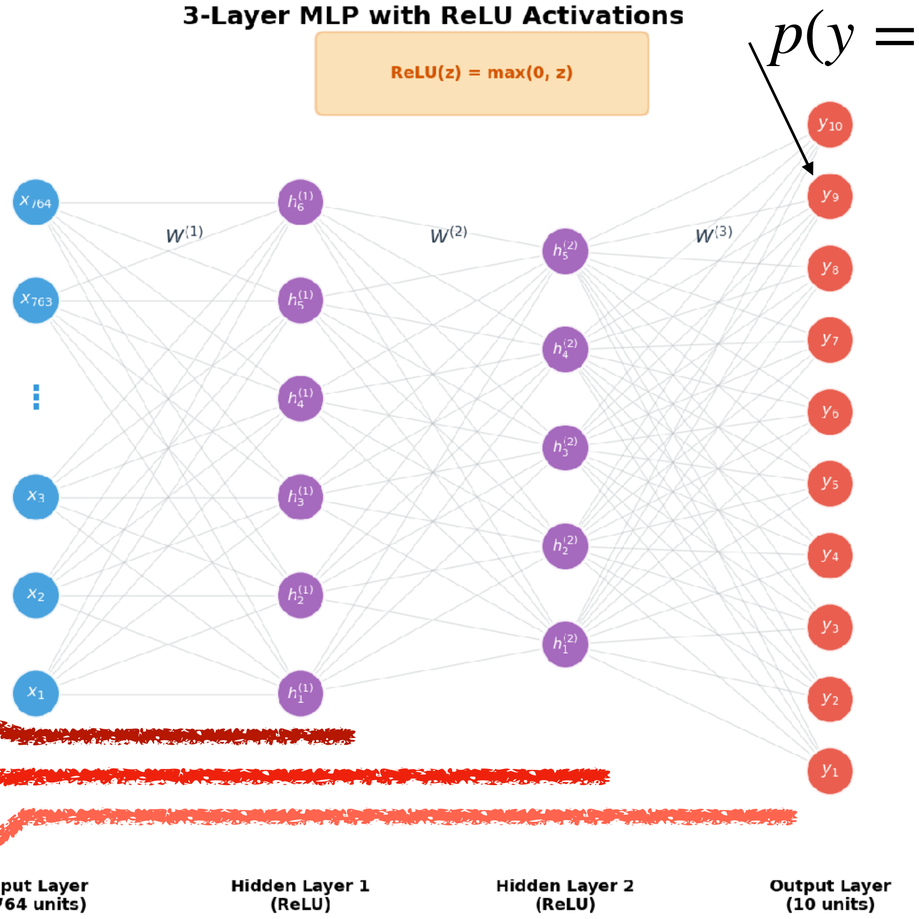
$$\mathbf{W}^{(3)}\text{ReLU}(\mathbf{W}^{(2)}\text{ReLU}(\mathbf{W}^{(1)}\mathbf{x}_n + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) + \mathbf{b}^{(3)}$$



How to get p(y=9|x_n)?

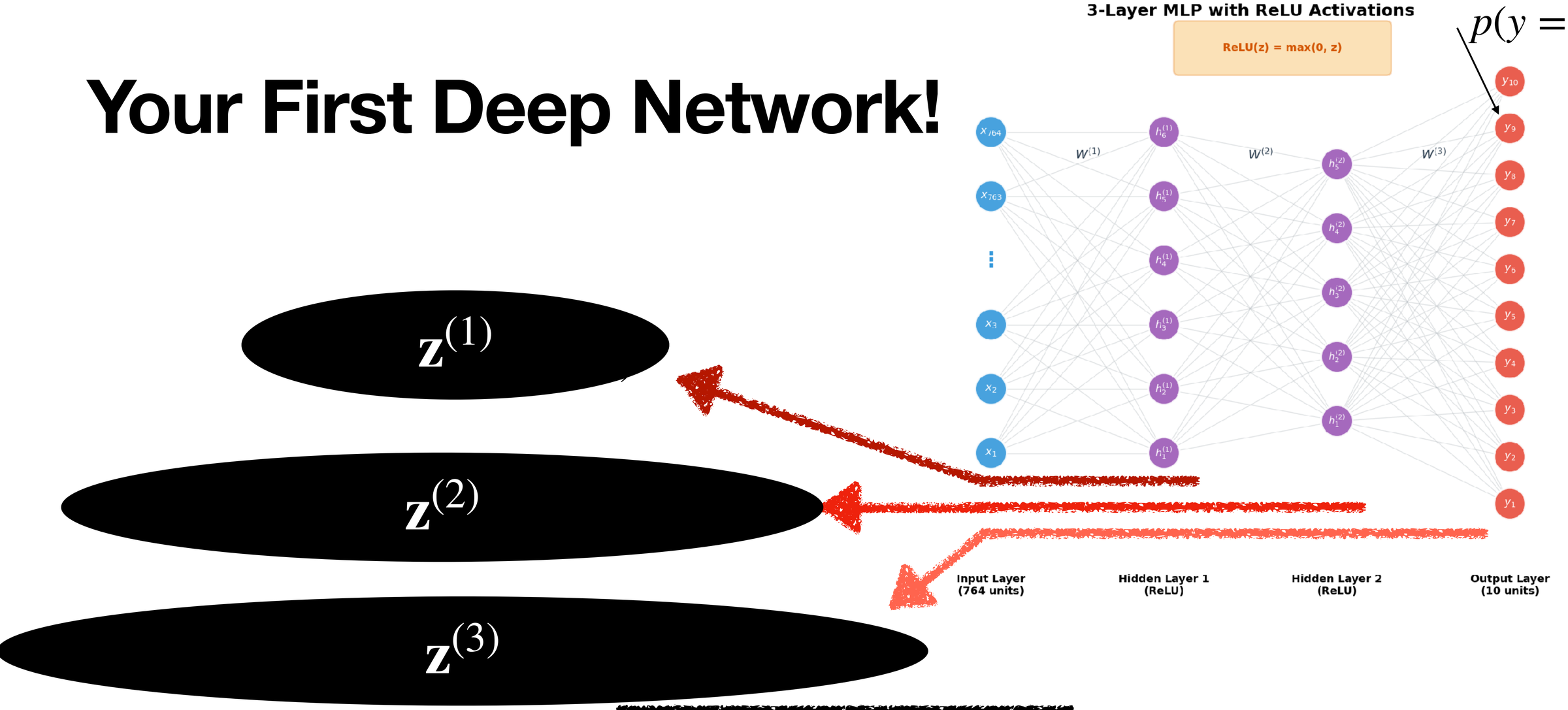
Your First Deep Network!





How to get p(y=9|x_n)?

Your First Deep Network!



$\mathbf{z}^{(1)}$

$\mathbf{z}^{(2)}$

$\mathbf{z}^{(3)}$

How to get p(y=9|x_n)?

$$p(y = c \mid \mathbf{x}_n) = \text{Softmax}(\mathbf{z}^{(3)})_c$$

The Softmax

- Think of it as taking any real vector and turning it into a probability vector

The Softmax

- Think of it as taking any real vector and turning it into a probability vector

$$\text{Softmax}(\mathbf{z}^{(3)})_c = \frac{e^{\mathbf{z}_c^{(3)}}}{\sum_{k=1}^{C} e^{\mathbf{z}_k^{(3)}}}$$

The Softmax

- Think of it as taking any real vector and turning it into a probability vector

$$\text{Softmax}(\mathbf{z}^{(3)})_c = \frac{e^{\mathbf{z}_c^{(3)}}}{\sum_{k=1}^{C} e^{\mathbf{z}_k^{(3)}}}$$

```
>>> def softmax(z):
...     return np.exp(z)/np.exp(z).sum()
...
```

The Softmax

- Think of it as taking any real vector and turning it into a probability vector

$$\text{Softmax}(\mathbf{z}^{(3)})_c = \frac{e^{\mathbf{z}_c^{(3)}}}{\sum_{k=1}^{C} e^{\mathbf{z}_k^{(3)}}}$$

```
>>> def softmax(z):
...     return np.exp(z)/np.exp(z).sum()
...
```

```
>>> softmax([-1,2,1.5])
array([0.03005889, 0.6037489 , 0.36619222])
```

The Softmax

- Think of it as taking any real vector and turning it into a probability vector

$$\text{Softmax}(\mathbf{z}^{(3)})_c = \frac{e^{\mathbf{z}_c^{(3)}}}{\sum_{k=1}^{C} e^{\mathbf{z}_k^{(3)}}}$$

```
>>> def softmax(z):
...     return np.exp(z)/np.exp(z).sum()
...
```

```
>>> softmax([-1,2,1.5])
array([0.03005889, 0.6037489 , 0.36619222])
```

```
>>> softmax([-1,2,1000.5])
```

The Softmax

- Think of it as taking any real vector and turning it into a probability vector

$$\text{Softmax}(\mathbf{z}^{(3)})_c = \frac{e^{\mathbf{z}_c^{(3)}}}{\sum_{k=1}^{C} e^{\mathbf{z}_k^{(3)}}}$$

```
>>> def softmax(z):
...     return np.exp(z)/np.exp(z).sum()
...
```

```
>>> softmax([-1,2,1.5])
array([0.03005889, 0.6037489 , 0.36619222])
```

```
>>> softmax([-1,2,1000.5])
<stdin>:2: RuntimeWarning: overflow encountered in exp
<stdin>:2: RuntimeWarning: invalid value encountered in divide
array([ 0.,  0., nan])
```

The Softmax

- Think of it as taking any real vector and turning it into a probability vector

The Softmax

- Think of it as taking any real vector and turning it into a probability vector

$$\text{Softmax}(\mathbf{z}^{(3)})_c = \frac{e^{\mathbf{z}_c^{(3)} - \max_i \mathbf{z}_i^{(3)}}}{\sum_{k=1}^{C} e^{\mathbf{z}_k^{(3)} - \max_i \mathbf{z}_i^{(3)}}}$$

The Softmax

- Think of it as taking any real vector and turning it into a probability vector

$$\text{Softmax}(\mathbf{z}^{(3)})_c = \frac{e^{\mathbf{z}_c^{(3)} - \max_i \mathbf{z}_i^{(3)}}}{\sum_{k=1}^{C} e^{\mathbf{z}_k^{(3)} - \max_i \mathbf{z}_i^{(3)}}}$$

- Our training loss becomes

The Softmax

- Think of it as taking any real vector and turning it into a probability vector

$$\text{Softmax}(\mathbf{z}^{(3)})_c = \frac{e^{\mathbf{z}_c^{(3)} - \max_i \mathbf{z}_i^{(3)}}}{\sum_{k=1}^{C} e^{\mathbf{z}_k^{(3)} - \max_i \mathbf{z}_i^{(3)}}}$$

- Our training loss becomes

$$\mathscr{L} = \sum_{n=1}^{N} - \sum_{c=1}^{10} 1_{\{y_n = c\}} \log(p(y = c \,|\, x_n)) = \sum_{n=1}^{N} - \sum_{n=1}^{N} \sum_{c=1}^{10} 1_{\{y_n = c\}} \log(\text{Softmax}(\mathbf{z}^{(3)}(\mathbf{x}_n))_c)$$

$$= \sum_{n=1}^{N} - \mathbf{z}^{(3)}(\mathbf{x}_n)_{y_n} + \log(\sum_{c=1}^{C} e^{\mathbf{z}^{(3)}(\mathbf{x}_n)_c})$$

The Softmax

- Think of it as taking any real vector and turning it into a probability vector

$$\text{Softmax}(\mathbf{z}^{(3)})_c = \frac{e^{\mathbf{z}^{(3)}_c - \max_i \mathbf{z}^{(3)}_i}}{\sum_{c=1}^{10} e^{\mathbf{z}^{(3)}_c - \max_i \mathbf{z}^{(3)}_i}}$$

- Our training loss

Fused op (cross_entropy + softmax)!!

$$\mathscr{L} = \sum_{n=1}^{N} - \sum_{c=1}^{10} 1_{\{y_n=c\}} \log(p(y=c\,|\,x_n)) = \sum_{n=1}^{N} - \sum_{n=1}^{N} \sum_{c=1}^{10} 1_{\{y_n=c\}} \log(\text{Softmax}(\mathbf{z}^{(3)}(\mathbf{x}_n))_c)$$

$$= \sum_{n=1}^{N} - \mathbf{z}^{(3)}(\mathbf{x}_n)_{y_n} + \log(\sum_{c=1}^{C} e^{\mathbf{z}^{(3)}(\mathbf{x}_n)_c})$$

The Softmax Gradient

- Remember: training involves $\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \lambda \nabla_{\mathbf{W}^{(l)}} \mathscr{L}, \forall l$

The Softmax Gradient

- Remember: training involves $\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \lambda \nabla_{\mathbf{W}^{(l)}} \mathscr{L}, \forall l$

$$\mathscr{L} = \sum_{n=1}^{N} - \mathbf{z}^{(3)}(\mathbf{x}_n)_{y_n} + \log(\sum_{c=1}^{C} e^{\mathbf{z}^{(3)}(\mathbf{x}_n)_c})$$

The Softmax Gradient

- Remember: training involves $\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \lambda \nabla_{\mathbf{W}^{(l)}} \mathscr{L}, \forall l$

$$\mathscr{L} = \sum_{n=1}^{N} - \mathbf{z}^{(3)}(\mathbf{x}_n)_{y_n} + \log(\sum_{c=1}^{C} e^{\mathbf{z}^{(3)}(\mathbf{x}_n)_c})$$

- There is no W here!!!

The Softmax Gradient

- Remember: training involves $\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \lambda \nabla_{\mathbf{W}^{(l)}} \mathscr{L}, \forall l$

$$\mathscr{L} = \sum_{n=1}^{N} - \mathbf{z}^{(3)}(\mathbf{x}_n)_{y_n} + \log(\sum_{c=1}^{C} e^{\mathbf{z}^{(3)}(\mathbf{x}_n)_c})$$

- There is no W here!!!

- Take the gradient of the loss w.r.t. $\mathbf{z}^{(3)}$, then multiply by the gradient of $\mathbf{z}^{(3)}$ w.r.t. $\mathbf{W}^{(3)}$ and so on!

The Softmax Gradient

- Denote by $\mathbf{v}_n$ the gradient of the loss w.r.t. $\mathbf{z}^{(3)}$

The Softmax Gradient

- Denote by $\mathbf{v}_n$ the gradient of the loss w.r.t. $\mathbf{z}^{(3)}$

$$\mathbf{W}^{(3)}\text{ReLU}(\mathbf{W}^{(2)}\text{ReLU}(\mathbf{W}^{(1)}\mathbf{x}_n + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) + \mathbf{b}^{(3)}$$

The Softmax Gradient

- Denote by $\mathbf{v}_n$ the gradient of the loss w.r.t. $\mathbf{z}^{(3)}$

$$\mathbf{W}^{(3)}\text{ReLU}(\mathbf{W}^{(2)}\text{ReLU}(\mathbf{W}^{(1)}\mathbf{x}_n + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) + \mathbf{b}^{(3)}$$

- What is the gradient of $\mathbf{z}^{(3)}$ w.r.t $\mathbf{z}^{(2)}$?

The Softmax Gradient

- Denote by $\mathbf{v}_n$ the gradient of the loss w.r.t. $\mathbf{z}^{(3)}$

$$\mathbf{W}^{(3)}\text{ReLU}(\mathbf{W}^{(2)}\text{ReLU}(\mathbf{W}^{(1)}\mathbf{x}_n + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) + \mathbf{b}^{(3)}$$

- What is the gradient of $\mathbf{z}^{(3)}$ w.r.t $\mathbf{z}^{(2)}$? $\mathbf{W}^{(3)}$

The Softmax Gradient

- Denote by $\mathbf{v}_n$ the gradient of the loss w.r.t. $\mathbf{z}^{(3)}$

$$\mathbf{W}^{(3)}\text{ReLU}(\mathbf{W}^{(2)}\text{ReLU}(\mathbf{W}^{(1)}\mathbf{x}_n + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) + \mathbf{b}^{(3)}$$

- What is the gradient of $\mathbf{z}^{(3)}$ w.r.t $\mathbf{z}^{(2)}$? $\mathbf{W}^{(3)}$
- What is the gradient of $\mathbf{z}^{(2)}$ w.r.t. $\mathbf{W}^{(2)}$?

The Softmax Gradient

- Denote by $\mathbf{v}_n$ the gradient of the loss w.r.t. $\mathbf{z}^{(3)}$

$$\mathbf{W}^{(3)}\text{ReLU}(\mathbf{W}^{(2)}\text{ReLU}(\mathbf{W}^{(1)}\mathbf{x}_n + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) + \mathbf{b}^{(3)}$$

- What is the gradient of $\mathbf{z}^{(3)}$ w.r.t $\mathbf{z}^{(2)}$? $\mathbf{W}^{(3)}$
- What is the gradient of $\mathbf{z}^{(2)}$ w.r.t. $\mathbf{W}^{(2)}$? $\mathbf{D}_n^{(2)}$

The Softmax Gradient

- Denote by $\mathbf{v}_n$ the gradient of the loss w.r.t. $\mathbf{z}^{(3)}$

$$\mathbf{W}^{(3)}\text{ReLU}(\mathbf{W}^{(2)}\text{ReLU}(\mathbf{W}^{(1)}\mathbf{x}_n + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) + \mathbf{b}^{(3)}$$

- What is the gradient of $\mathbf{z}^{(3)}$ w.r.t $\mathbf{z}^{(2)}$? $\mathbf{W}^{(3)}$
- What is the gradient of $\mathbf{z}^{(2)}$ w.r.t. $\mathbf{W}^{(2)}$? $\mathbf{D}_n^{(2)}$
- What is the gradient of the loss w.r.t. $\mathbf{W}^{(2)}$?

The Softmax Gradient

- Denote by $\mathbf{v}_n$ the gradient of the loss w.r.t. $\mathbf{z}^{(3)}$

$$\mathbf{W}^{(3)}\text{ReLU}(\mathbf{W}^{(2)}\text{ReLU}(\mathbf{W}^{(1)}\mathbf{x}_n + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) + \mathbf{b}^{(3)}$$

- What is the gradient of $\mathbf{z}^{(3)}$ w.r.t $\mathbf{z}^{(2)}$? $\mathbf{W}^{(3)}$
- What is the gradient of $\mathbf{z}^{(2)}$ w.r.t. $\mathbf{W}^{(2)}$? $\mathbf{D}_n^{(2)}$
- What is the gradient of the loss w.r.t. $\mathbf{W}^{(2)}$? $(\mathbf{v}_n^\top \mathbf{W}^{(3)} \mathbf{D}_n^{(2)})^\top \mathbf{z}^{(1)}(\mathbf{x}_n)$

The Full Story

The Full Story

- Compute the loss from your samples $\{(\mathbf{x}_n, \mathbf{y}_n), n = 1, \ldots, N\}$

The Full Story

- Compute the loss from your samples $\{(\mathbf{x}_n, \mathbf{y}_n), n = 1, \ldots, N\}$
- Compute the gradients via chain rule

The Full Story

- Compute the loss from your samples $\{(\mathbf{x}_n, \mathbf{y}_n), n = 1, \ldots, N\}$
- Compute the gradients via chain rule
- Update your parameters

The Full Story

- Compute the loss from your samples $\{(\mathbf{x}_n, \mathbf{y}_n), n = 1, \ldots, N\}$
- Compute the gradients via chain rule
- Update your parameters
- Repeat

The Full Story

- Compute the loss from your samples $\{(\mathbf{x}_n, \mathbf{y}_n), n = 1, \ldots, N\}$
- Compute the gradients via chain rule
- Update your parameters
- Repeat

Can you think of an improvement?

The Full Story

The Full Story

- Sample a mini-batch $\{(\mathbf{x}_i, \mathbf{y}_i), i \in \mathscr{B}\}$ with $\mathscr{B} \subset \{1, \ldots, N\}$ a random subset

The Full Story

- Sample a mini-batch $\{(\mathbf{x}_i, \mathbf{y}_i), i \in \mathscr{B}\}$ with $\mathscr{B} \subset \{1, \ldots, N\}$ a random subset
- Compute the loss from your mini-batch $\{(\mathbf{x}_i, \mathbf{y}_i), i \in \mathscr{B}\}$

The Full Story

- Sample a mini-batch $\{(\mathbf{x}_i, \mathbf{y}_i), i \in \mathscr{B}\}$ with $\mathscr{B} \subset \{1, \ldots, N\}$ a random subset
- Compute the loss from your mini-batch $\{(\mathbf{x}_i, \mathbf{y}_i), i \in \mathscr{B}\}$
- Compute the gradients via chain rule

The Full Story

- Sample a mini-batch $\{(\mathbf{x}_i, \mathbf{y}_i), i \in \mathscr{B}\}$ with $\mathscr{B} \subset \{1, \ldots, N\}$ a random subset
- Compute the loss from your mini-batch $\{(\mathbf{x}_i, \mathbf{y}_i), i \in \mathscr{B}\}$
- Compute the gradients via chain rule
- Update your parameters

The Full Story

- Sample a mini-batch $\{(\mathbf{x}_i, \mathbf{y}_i), i \in \mathscr{B}\}$ with $\mathscr{B} \subset \{1, \ldots, N\}$ a random subset
- Compute the loss from your mini-batch $\{(\mathbf{x}_i, \mathbf{y}_i), i \in \mathscr{B}\}$
- Compute the gradients via chain rule
- Update your parameters
- Repeat

The Full Story

- Sample a mini-batch $\{(\mathbf{x}_i, \mathbf{y}_i), i \in \mathcal{B}\}$ with $\mathcal{B} \subset \{1, \ldots, N\}$ a random subset
- Compute the loss from your mini-batch $\{(\mathbf{x}_i, \mathbf{y}_i), i \in \mathcal{B}\}$
- Compute the gradients via chain rule
- Update your parameters
- Repeat

How to pick the mini-batch size?

Beyond Gradient Descent

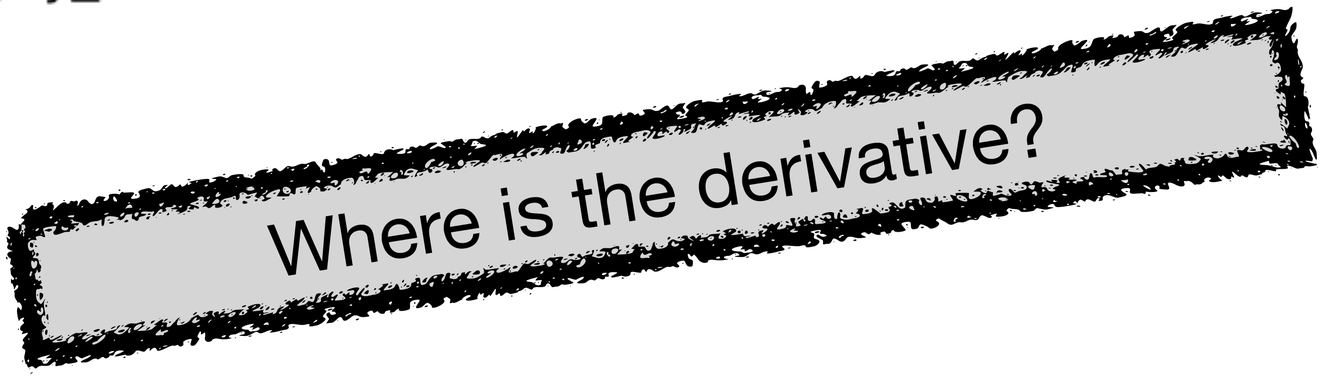
Optimizer	Update idea (per step)	Strengths	Weaknesses / Notes
SGD	`θ ← θ − lr ∗ g`	Simple, stable; often good generalization	Requires tuning `lr`; can be slow without momentum
SGD + Momentum	`v ← μv + g`; `θ ← θ − lr ∗ v`	Faster convergence; smooths noisy gradients	Extra hyperparameter (momentum μ)
SGD + Nesterov	Look ahead before gradient step	Often converges faster than vanilla momentum	Slightly more complex; similar tuning issues
Adam	Per-parameter adaptive lr using 1st & 2nd moment estimates	Usually works "out of the box"; good for sparse grads	Can generalize worse than SGD; more hyperparameters
AdamW	Adam + *decoupled* weight decay	Better weight decay behavior; often preferred default	Slightly more to configure (weight_decay)
RMSprop	Scales lr by running avg of squared grads	Good for non-stationary problems; common in RNNs	Can be sensitive to hyperparameters
Adagrad	Accumulates squared grads, shrinking lr over time	Good for sparse features; no manual lr schedule	Learning rate can become too small over long training

The Full (real) Story

```
for x_batch, y_batch in loader:
    optimizer.zero_grad()
    # Forward pass: raw logits (no softmax)
    logits = model(x_batch)
    # Functional cross entropy:
    # takes logits and class indices directly
    loss = F.cross_entropy(logits, y_batch)
    loss.backward()
    optimizer.step()
```

The Full (real) Story

```
for x_batch, y_batch in loader:
    optimizer.zero_grad()
    # Forward pass: raw logits (no softmax)
    logits = model(x_batch)
    # Functional cross entropy:
    # takes logits and class indices directly
    loss = F.cross_entropy(logits, y_batch)
    loss.backward()
    optimizer.step()
```



Autodiff!

- All deep learning frameworks implement automatic differentiation

Autodiff!

- All deep learning frameworks implement automatic differentiation

Autodiff!

- All deep learning frameworks implement automatic differentiation

Autodiff!

- All deep learning frameworks implement automatic differentiation

Autodiff!

- All deep learning frameworks implement automatic differentiation

Autodiff!

- All deep learning frameworks implement automatic differentiation

- You still need to know about chain-rule and gradients!

Autodiff!

- All deep learning frameworks implement automatic differentiation

- You still need to know about chain-rule and gradients!
- The efficient implementation is called “backdrop” with vjp (or jvp)

The Elephant in the Room

The Elephant in the Room

The Elephant in the Room

The Elephant in the Room

The Elephant in the Room

The Elephant in the Room

- Which framework to use?

The Elephant in the Room

- Which framework to use?

What are our hyper-parameters so far?

- Preprocessing of the data
- Number of layers (L), width of each layer
- Initialization of the parameters
- Optimizer, learning rate (+ momentum, …)
- Mini-batch size
- Training steps

Initialization Brainstorming

$$\mathbf{W}^{(3)}\text{ReLU}(\mathbf{W}^{(2)}\text{ReLU}(\mathbf{W}^{(1)}\mathbf{x}_n + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) + \mathbf{b}^{(3)}$$

Initialization Brainstorming

$$\mathbf{W}^{(3)}\text{ReLU}(\mathbf{W}^{(2)}\text{ReLU}(\mathbf{W}^{(1)}\mathbf{x}_n + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) + \mathbf{b}^{(3)}$$

What would be a good/bad initialization?