

Lecture 11

Gradient Descent and Non-Linear Function Approximation

(Neural Networks)

Last Time

- Rejection Sampling (Steroids) or with majorization
- Logistic Regression and Gradient Descent
- Stochastic Gradient Descent (simple)
- Importance Sampling and expectations

Today

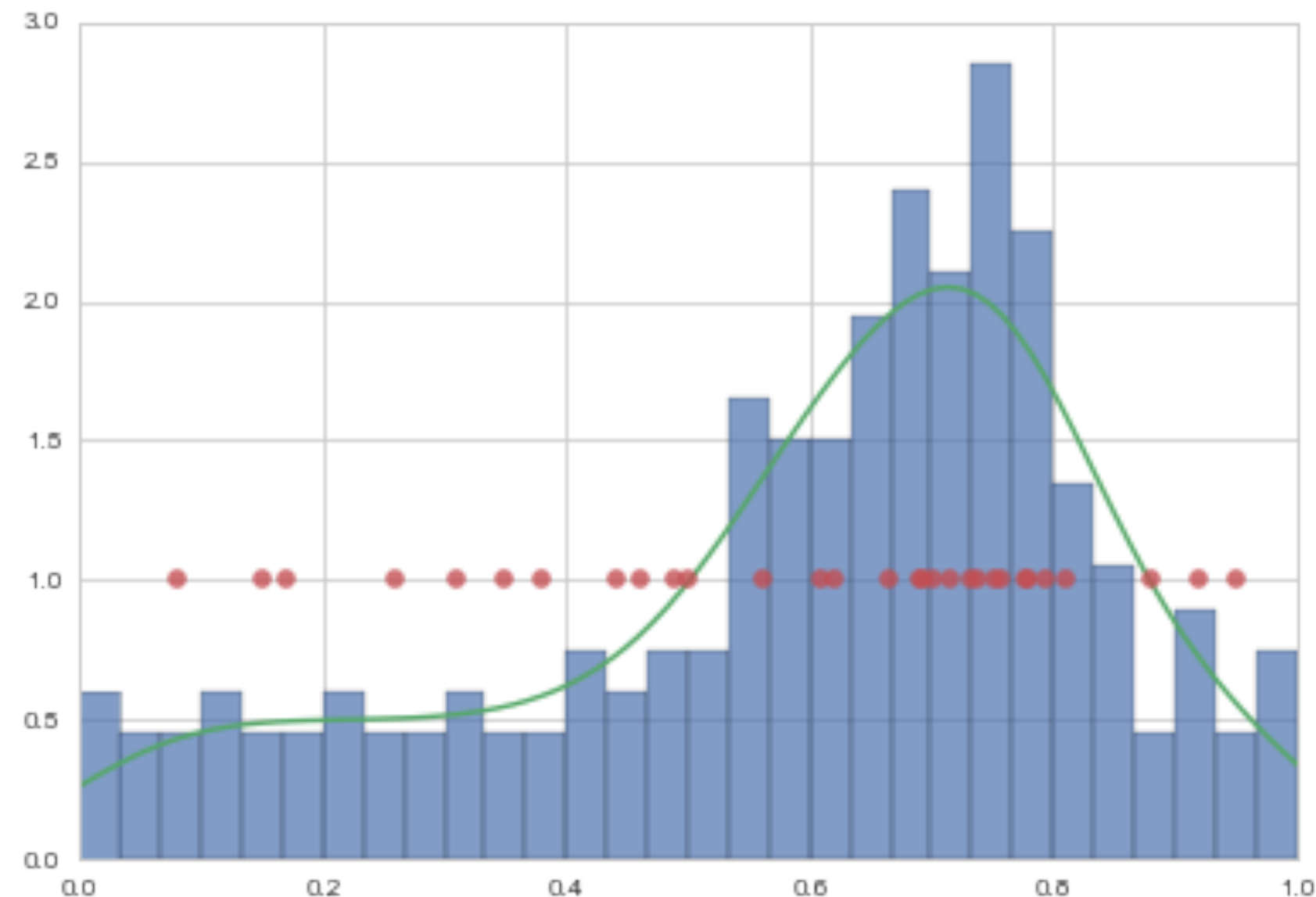
- More Logistic Regression: arranging in layers
- Reverse Mode Differentiation
- A general way of solving SGD problems
- Neural Networks
- SGD and linear models: Universal Approximation

Statement of the Learning Problem

The sample must be representative of the population!

$$\begin{aligned} A : R_{\mathcal{D}}(g) \text{ smallest on } \mathcal{H} \\ B : R_{out}(g) \approx R_{\mathcal{D}}(g) \end{aligned}$$

A: Empirical risk estimates in-sample risk.
B: Thus the out of sample risk is also small.



What we'd really like: population

i.e. out of sample RISK

$$\langle R_{out} \rangle = E_{p(x,y)} [R(h(x), y)] = \int dy dx p(x, y) R(h(x), y)$$

- But we only have the in-sample risk, furthermore its an empirical risk
- And its not even a full on empirical distribution, as N is usually quite finite

LLN, again

The sample empirical distribution converges to the true population distribution as $N \rightarrow \infty$

Then we'll want an average over possible samples generated from the population.

We don't have that, so we:

- stick to empirical risk in one sample, but then
- engage in train-test, validation, and cross-validation in our sample

Gradient Descent.

For a particular sample, we want:

$$\nabla_h R_{out}(h, y) = \int dx p(x) \nabla_h R_{out}(h(x), y) (e.g.).$$

$$\text{LLN:} = \nabla_h \frac{1}{N} \sum_{i \in \text{pop}} R_{out}(h(x_i), y_i) \sim \nabla_h \frac{1}{N} \sum_{i \in \mathcal{D}} R_{in}(h(x_i), y_i)$$

Gradient Descent

$$\theta := \theta - \eta \nabla_{\theta} R(\theta) = \theta - \eta \sum_{i=1}^m \nabla R_i(\theta)$$

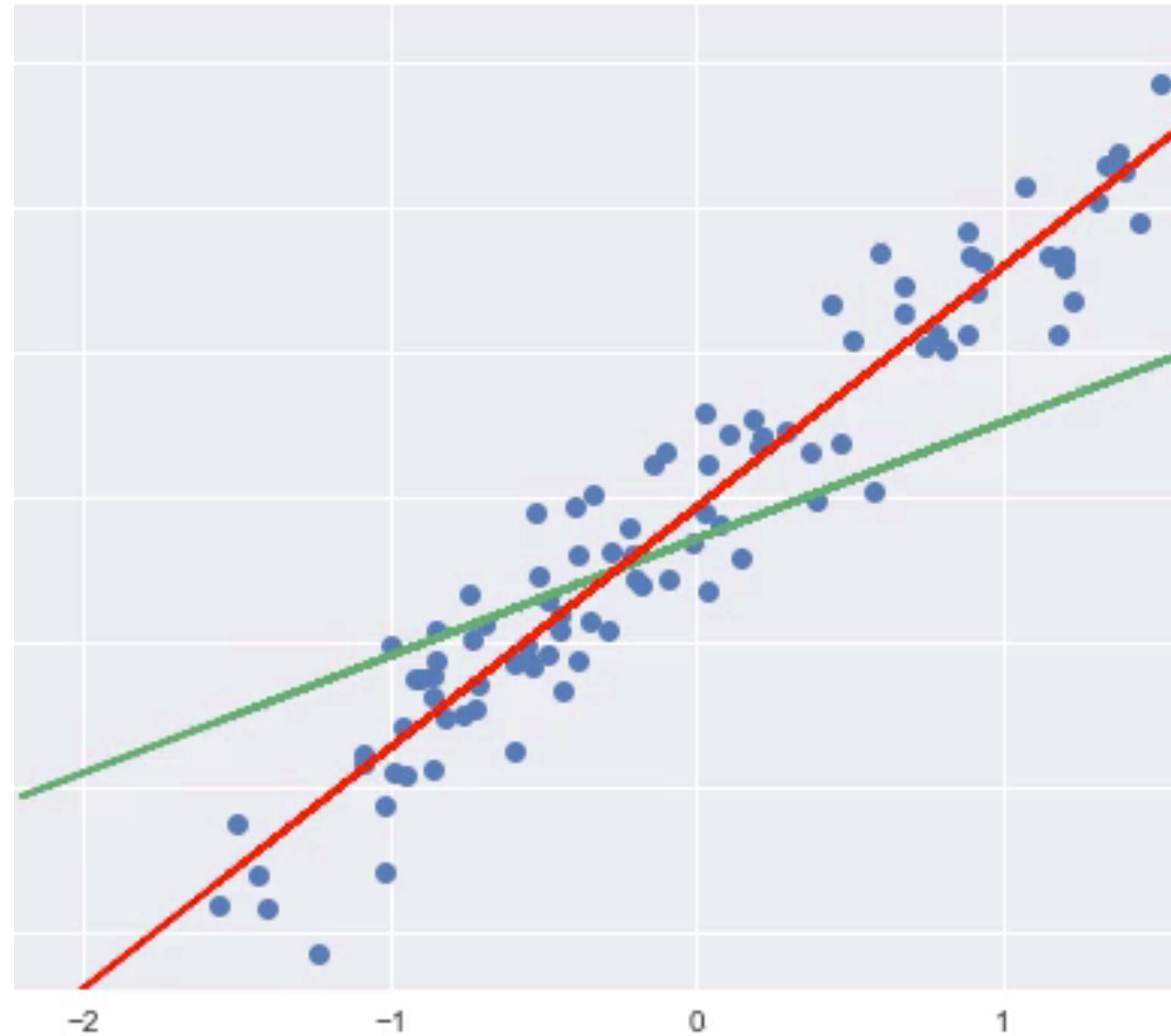
where η is the learning rate.

ENTIRE DATASET NEEDED

```
for i in range(n_epochs):  
    params_grad = evaluate_gradient(loss_function, data, params)  
    params = params - learning_rate * params_grad`
```


Linear Regression: Gradient Descent

$$\theta_j := \theta_j + \alpha \sum_{i=1}^m (y^{(i)} - f_{\theta}(x^{(i)})) x_j^{(i)}$$



Stochastic Gradient Descent

$$\theta := \theta - \alpha \nabla_{\theta} R_i(\theta)$$

ONE POINT AT A TIME

For Linear Regression:

$$\theta_j := \theta_j + \alpha(y^{(i)} - f_{\theta}(x^{(i)}))x_j^{(i)}$$

```
for i in range(nb_epochs):  
    np.random.shuffle(data)  
    for example in data:  
        params_grad = evaluate_gradient(loss_function, example, params)  
        params = params - learning_rate * params_grad
```

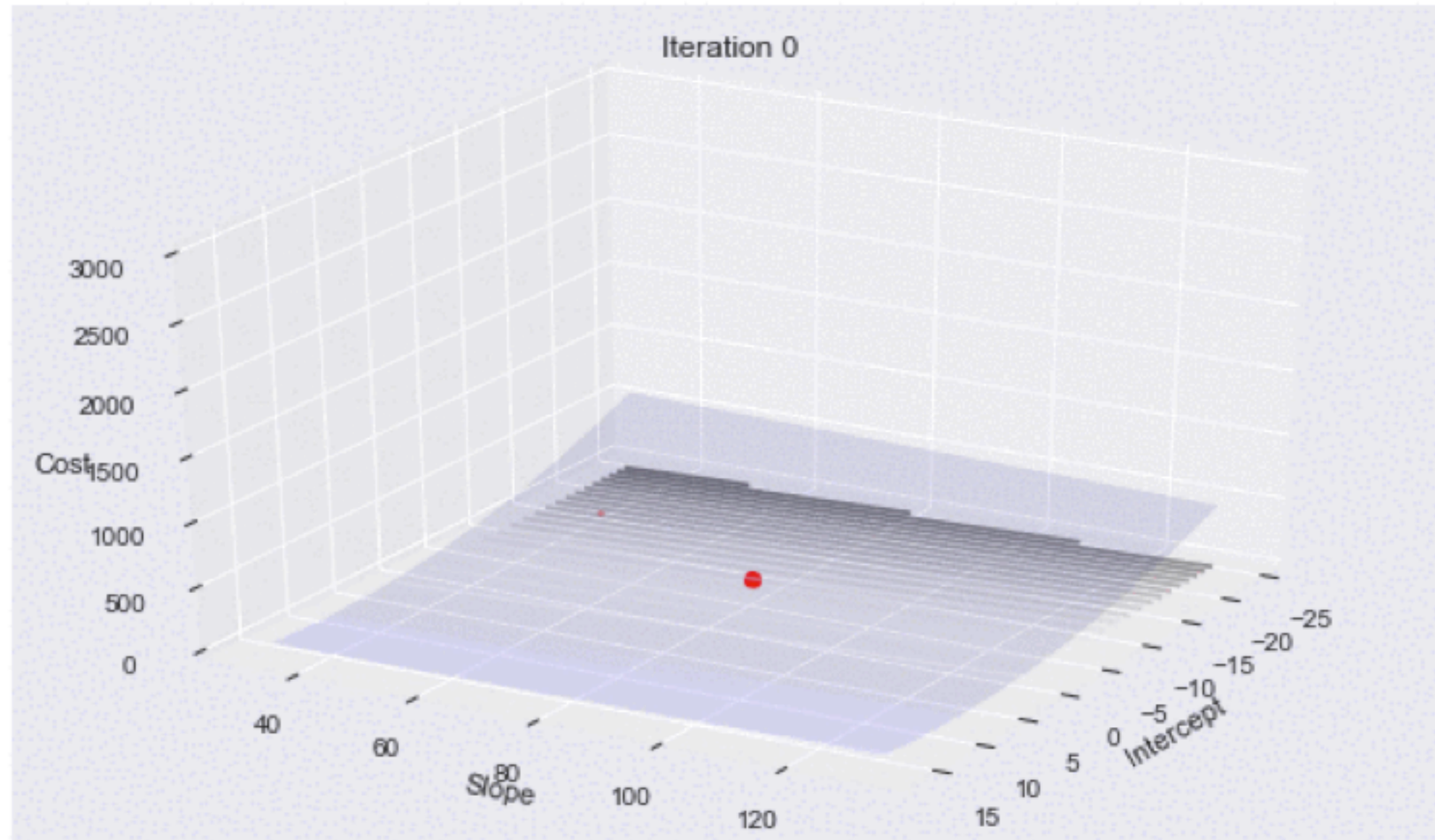
Mini-Batch SGD (the most used)

$$\theta := \theta - \eta \nabla_{\theta} J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

```
for i in range(mb_epochs):  
    np.random.shuffle(data)  
    for batch in get_batches(data, batch_size=50):  
        params_grad = evaluate_gradient(loss_function, batch, params)  
        params = params - learning_rate * params_grad
```

Mini-Batch: do some at a time

- the risk surface changes at each gradient calculation
- thus things are noisy
- cumulated risk is smoother, can be used to compare to SGD
- epochs are now the number of times you revisit the full dataset
- shuffle in-between to provide even more stochasticity



MLE for Logistic Regression

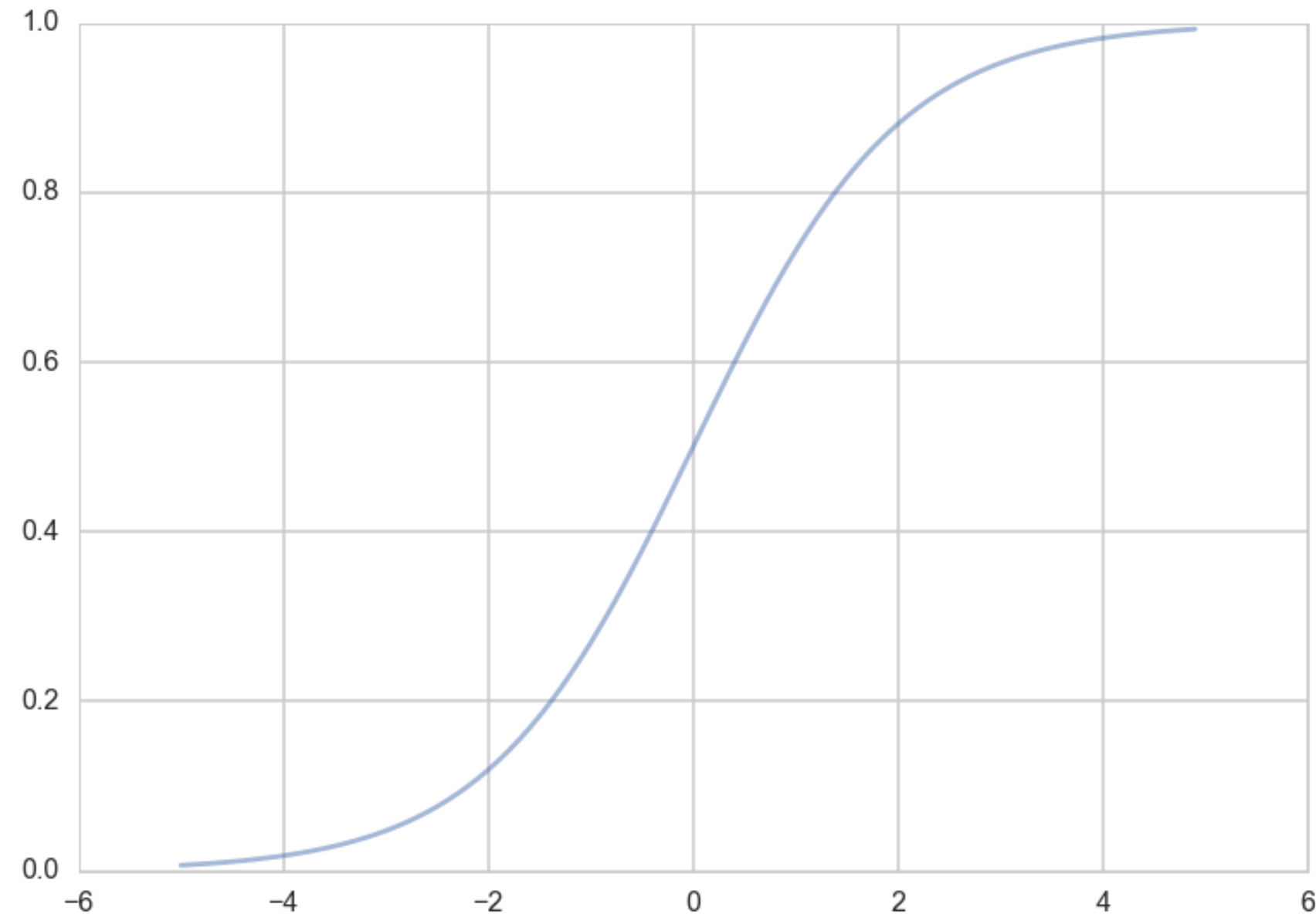
- example of a Generalized Linear Model (GLM)
- "Squeeze" linear regression through a **Sigmoid** function
- this bounds the output to be a probability
- What is the sampling Distribution?

Sigmoid function

This function is plotted below:

```
h = lambda z: 1./(1+np.exp(-z))  
zs=np.arange(-5,5,0.1)  
plt.plot(zs, h(zs), alpha=0.5);
```

Identify: $z = \mathbf{w} \cdot \mathbf{x}$ and $h(\mathbf{w} \cdot \mathbf{x})$ with the probability that the sample is a '1' ($y = 1$).



Then, the conditional probabilities of $y = 1$ or $y = 0$ given a particular sample's features \mathbf{x} are:

$$P(y = 1|\mathbf{x}) = h(\mathbf{w} \cdot \mathbf{x})$$
$$P(y = 0|\mathbf{x}) = 1 - h(\mathbf{w} \cdot \mathbf{x}).$$

These two can be written together as

$$P(y|\mathbf{x}, \mathbf{w}) = h(\mathbf{w} \cdot \mathbf{x})^y (1 - h(\mathbf{w} \cdot \mathbf{x}))^{(1-y)}$$

BERNOULLI!!

Multiplying over the samples we get:

$$P(y|\mathbf{x}, \mathbf{w}) = P(\{y_i\}|\{\mathbf{x}_i\}, \mathbf{w}) = \prod_{y_i \in \mathcal{D}} P(y_i|\mathbf{x}_i, \mathbf{w}) = \prod_{y_i \in \mathcal{D}} h(\mathbf{w} \cdot \mathbf{x}_i)^{y_i} (1 - h(\mathbf{w} \cdot \mathbf{x}_i))^{(1-y_i)}$$

Indeed its important to realize that a particular sample can be thought of as a draw from some "true" probability distribution.

maximum likelihood estimation maximises the **likelihood of the sample \mathbf{y}** , or alternately the log-likelihood,

$$\mathcal{L} = P(y \mid \mathbf{x}, \mathbf{w}). \text{ OR } \ell = \log(P(y \mid \mathbf{x}, \mathbf{w}))$$

Thus

$$\begin{aligned}\ell &= \log \left(\prod_{y_i \in \mathcal{D}} h(\mathbf{w} \cdot \mathbf{x}_i)^{y_i} (1 - h(\mathbf{w} \cdot \mathbf{x}_i))^{(1-y_i)} \right) \\&= \sum_{y_i \in \mathcal{D}} \log \left(h(\mathbf{w} \cdot \mathbf{x}_i)^{y_i} (1 - h(\mathbf{w} \cdot \mathbf{x}_i))^{(1-y_i)} \right) \\&= \sum_{y_i \in \mathcal{D}} \log h(\mathbf{w} \cdot \mathbf{x}_i)^{y_i} + \log (1 - h(\mathbf{w} \cdot \mathbf{x}_i))^{(1-y_i)} \\&= \sum_{y_i \in \mathcal{D}} (y_i \log(h(\mathbf{w} \cdot \mathbf{x})) + (1 - y_i) \log(1 - h(\mathbf{w} \cdot \mathbf{x})))\end{aligned}$$

Logistic Regression: NLL

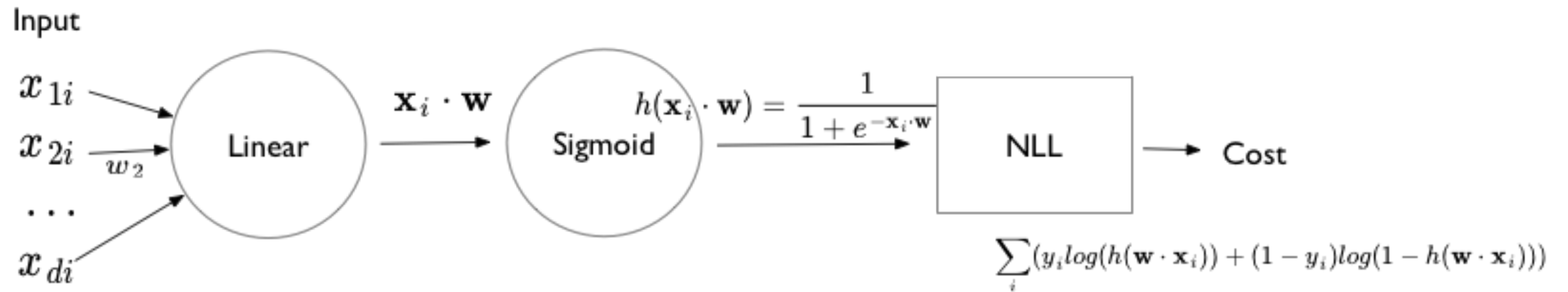
The negative of this log likelihood (NLL), also called *cross-entropy*.

$$NLL = - \sum_{y_i \in \mathcal{D}} (y_i \log(h(\mathbf{w} \cdot \mathbf{x})) + (1 - y_i) \log(1 - h(\mathbf{w} \cdot \mathbf{x})))$$

$$\text{Gradient: } \nabla_{\mathbf{w}} NLL = \sum_i \mathbf{x}_i^T (p_i - y_i) = \mathbf{X}^T \cdot (\mathbf{p} - \mathbf{w})$$

$$\text{Hessian: } H = \mathbf{X}^T \text{diag}(p_i(1 - p_i))\mathbf{X} \text{ positive definite} \implies \text{convex}$$

Units based diagram



Softmax formulation

- Identify p_i and $1 - p_i$ as two separate probabilities constrained to add to 1. That is $p_{1i} = p_i; p_{2i} = 1 - p_i$.
- $$p_{1i} = \frac{e^{\mathbf{w}_1 \cdot \mathbf{x}}}{e^{\mathbf{w}_1 \cdot \mathbf{x}} + e^{\mathbf{w}_2 \cdot \mathbf{x}}}$$
- $$p_{2i} = \frac{e^{\mathbf{w}_2 \cdot \mathbf{x}}}{e^{\mathbf{w}_1 \cdot \mathbf{x}} + e^{\mathbf{w}_2 \cdot \mathbf{x}}}$$
- Can translate coefficients by fixed amount ψ without any change

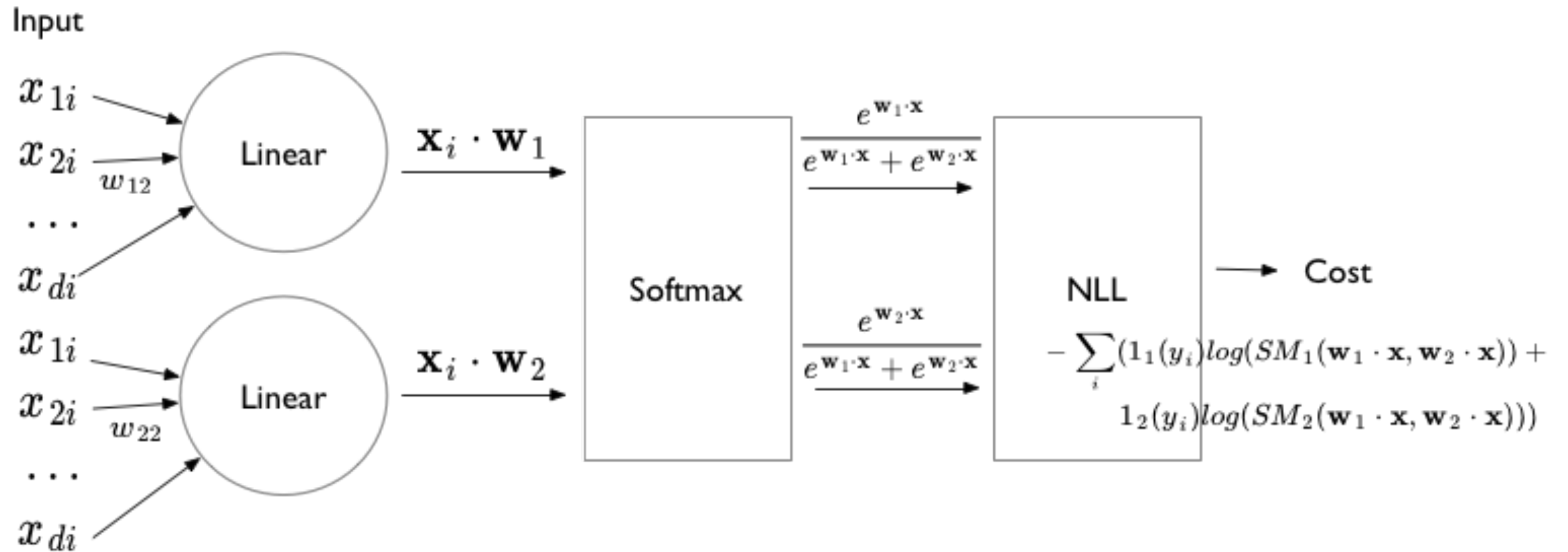
NLL and gradients for Softmax

$$\mathcal{L} = \prod_i p_{1i}^{1_1(y_i)} p_{2i}^{1_2(y_i)}$$

$$NLL = - \sum_i (1_1(y_i) \log(p_{1i}) + 1_2(y_i) \log(p_{2i}))$$

$$\frac{\partial NLL}{\partial \mathbf{w}_1} = - \sum_i \mathbf{x}_i (y_i - p_{1i}), \quad \frac{\partial NLL}{\partial \mathbf{w}_2} = - \sum_i \mathbf{x}_i (y_i - p_{2i})$$

Units diagram for Softmax

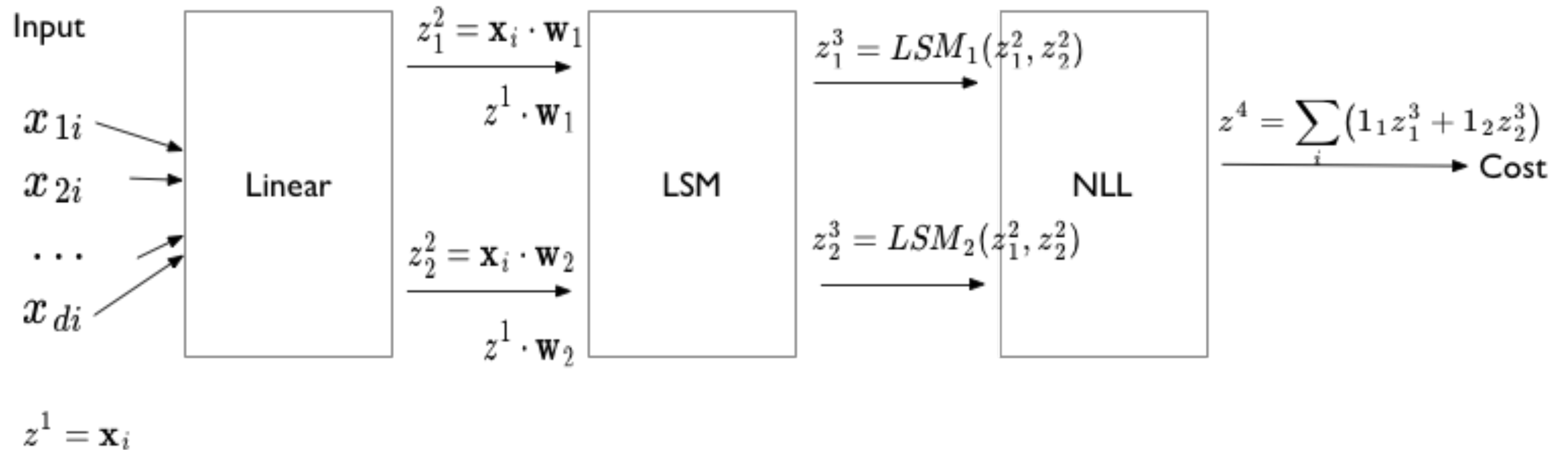


Rewrite NLL

$$NLL = - \sum_i (1_1(y_i) LSM_1(\mathbf{w}_1 \cdot \mathbf{x}, \mathbf{w}_2 \cdot \mathbf{x}) + 1_2(y_i) LSM_2(\mathbf{w}_1 \cdot \mathbf{x}, \mathbf{w}_2 \cdot \mathbf{x}))$$

where $SM_1 = \frac{e^{\mathbf{w}_1 \cdot \mathbf{x}}}{e^{\mathbf{w}_1 \cdot \mathbf{x}} + e^{\mathbf{w}_2 \cdot \mathbf{x}}}$ puts the first argument in the numerator. Ditto for LSM_1 which is simply $\log(SM_1)$.

Units diagram Again



Equations, layer by layer

$$\mathbf{z}^1 = \mathbf{x}_i$$

$$\mathbf{z}^2 = (z_1^2, z_2^2) = (\mathbf{w}_1 \cdot \mathbf{x}_i, \mathbf{w}_2 \cdot \mathbf{x}_i) = (\mathbf{w}_1 \cdot \mathbf{z}_i^1, \mathbf{w}_2 \cdot \mathbf{z}_i^1)$$

$$\mathbf{z}^3 = (z_1^3, z_2^3) = (LSM_1(z_1^2, z_2^2), LSM_2(z_1^2, z_2^2))$$

$$z^4 = NLL(\mathbf{z}^3) = NLL(z_1^3, z_2^3) = - \sum_i (1_1(y_i) z_1^3(i) + 1_2(y_i) z_2^3(i))$$

Reverse Mode Differentiation

$$Cost = f^{Loss}(\mathbf{f}^3(\mathbf{f}^2(\mathbf{f}^1(\mathbf{x}))))$$

$$\nabla_{\mathbf{x}} Cost = \frac{\partial f^{Loss}}{\partial \mathbf{f}^3} \frac{\partial \mathbf{f}^3}{\partial \mathbf{f}^2} \frac{\partial \mathbf{f}^2}{\partial \mathbf{f}^1} \frac{\partial \mathbf{f}^1}{\partial \mathbf{x}}$$

Write as:

$$\nabla_{\mathbf{x}} Cost = \left(\left(\left(\frac{\partial f^{Loss}}{\partial \mathbf{f}^3} \frac{\partial \mathbf{f}^3}{\partial \mathbf{f}^2} \right) \frac{\partial \mathbf{f}^2}{\partial \mathbf{f}^1} \right) \frac{\partial \mathbf{f}^1}{\partial \mathbf{x}} \right)$$

in digit classification

1x10

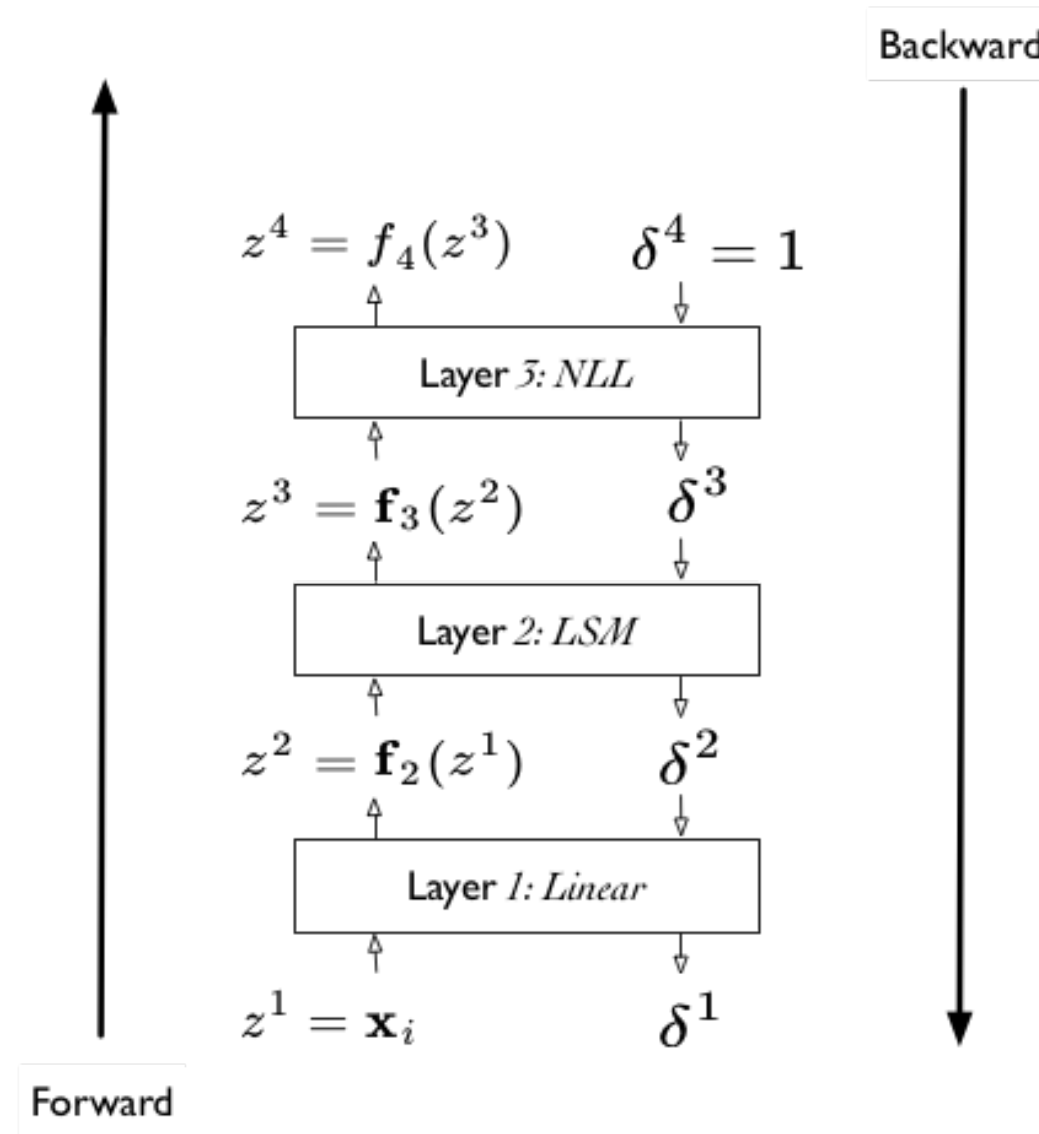
10x10

10x10

From Reverse Mode to Back Propagation

- Recursive Structure
- Always a vector times a Jacobian
- We add a "cost layer" to z^4 . The derivative of this layer with respect to z^4 will always be 1.
- We then propagate this derivative back.

Layer Cake



Backpropagation

RULE1: FORWARD (`.forward` in pytorch) $\mathbf{z}^{l+1} = \mathbf{f}^l(\mathbf{z}^l)$

RULE2: BACKWARD (`.backward` in pytorch)

$$\delta^l = \frac{\partial C}{\partial \mathbf{z}^l} \text{ or } \delta_u^l = \frac{\partial C}{\partial z_u^l}.$$

$$\delta_u^l = \frac{\partial C}{\partial z_u^l} = \sum_v \frac{\partial C}{\partial z_v^{l+1}} \frac{\partial z_v^{l+1}}{\partial z_u^l} = \sum_v \delta_v^{l+1} \frac{\partial z_v^{l+1}}{\partial z_u^l}$$

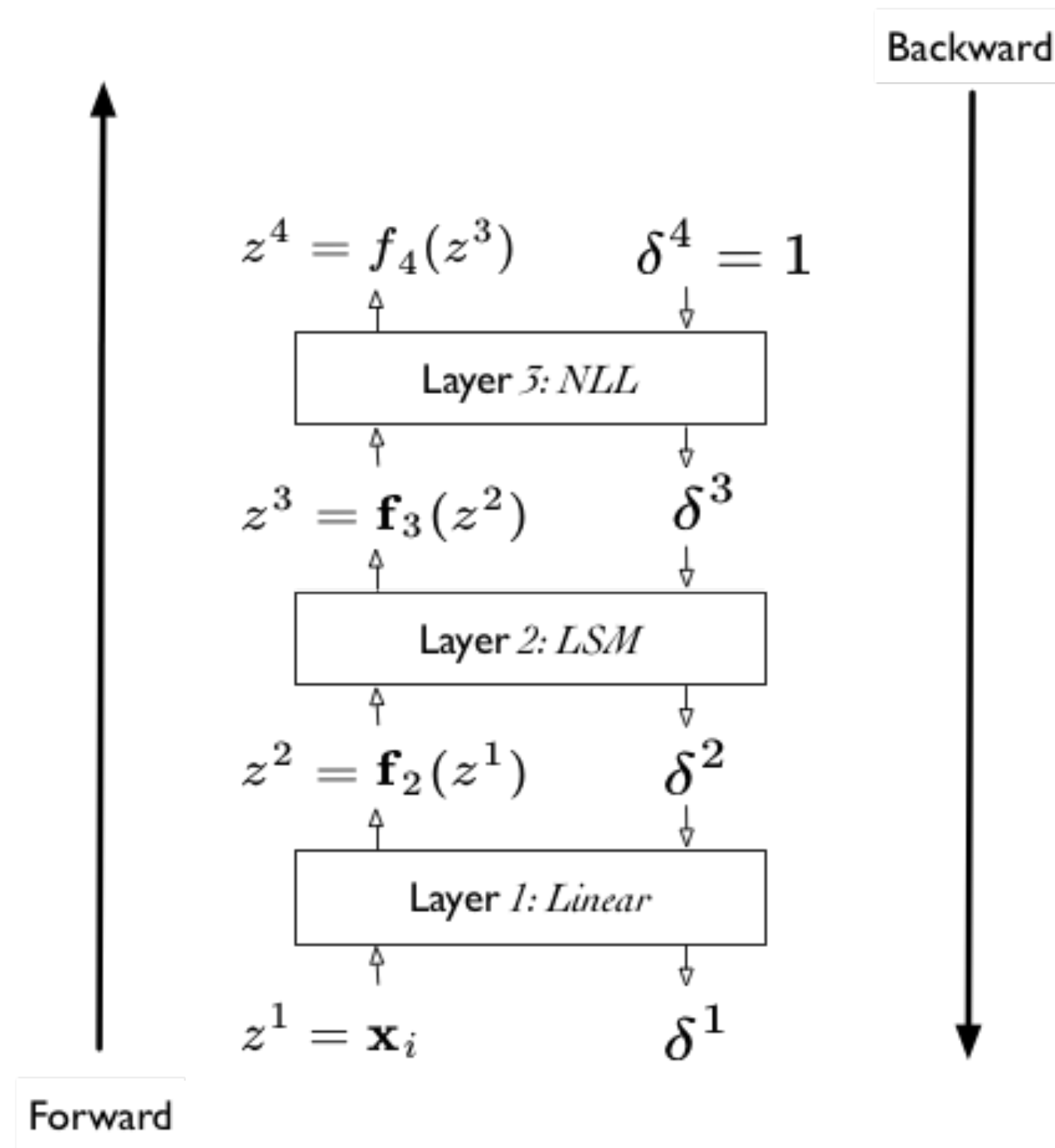
In particular:

$$\delta_u^3 = \frac{\partial z^4}{\partial z_u^3} = \frac{\partial C}{\partial z_u^3}$$

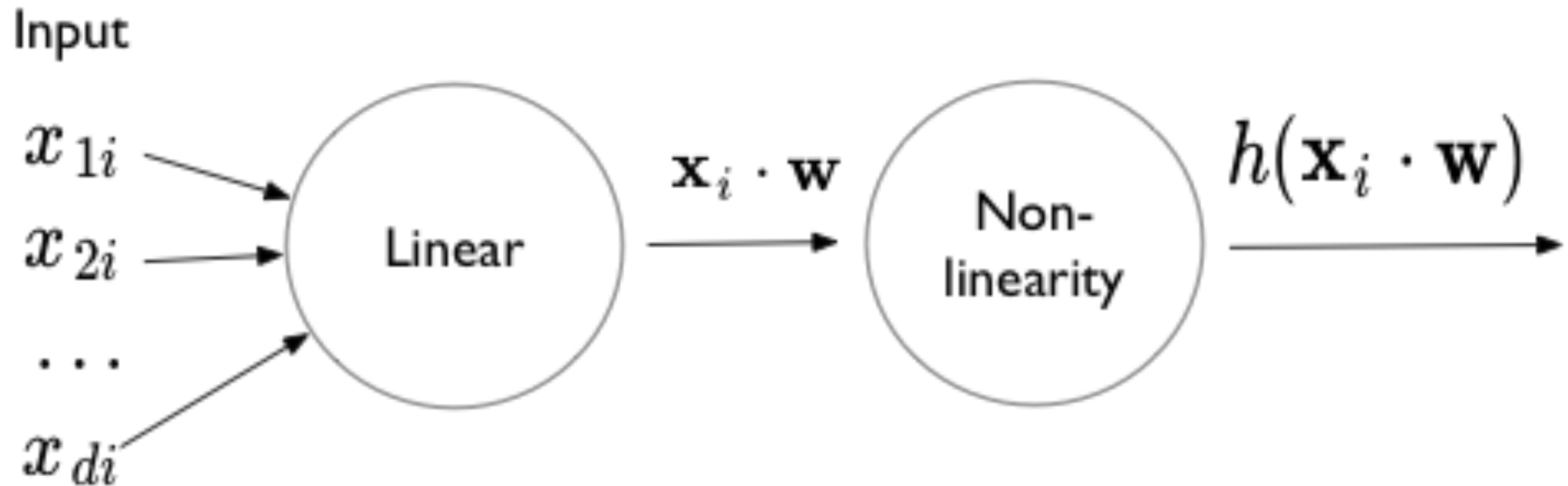
RULE 3: PARAMETERS

$$\frac{\partial C}{\partial \theta^l} = \sum_u \frac{\partial C}{\partial z_u^{l+1}} \frac{\partial z_u^{l+1}}{\partial \theta^l} = \sum_u \delta_u^{l+1} \frac{\partial z_u^{l+1}}{\partial \theta^l}$$

(backward pass is thus also used to fill the `variable.grad` parts of parameters in pytorch)



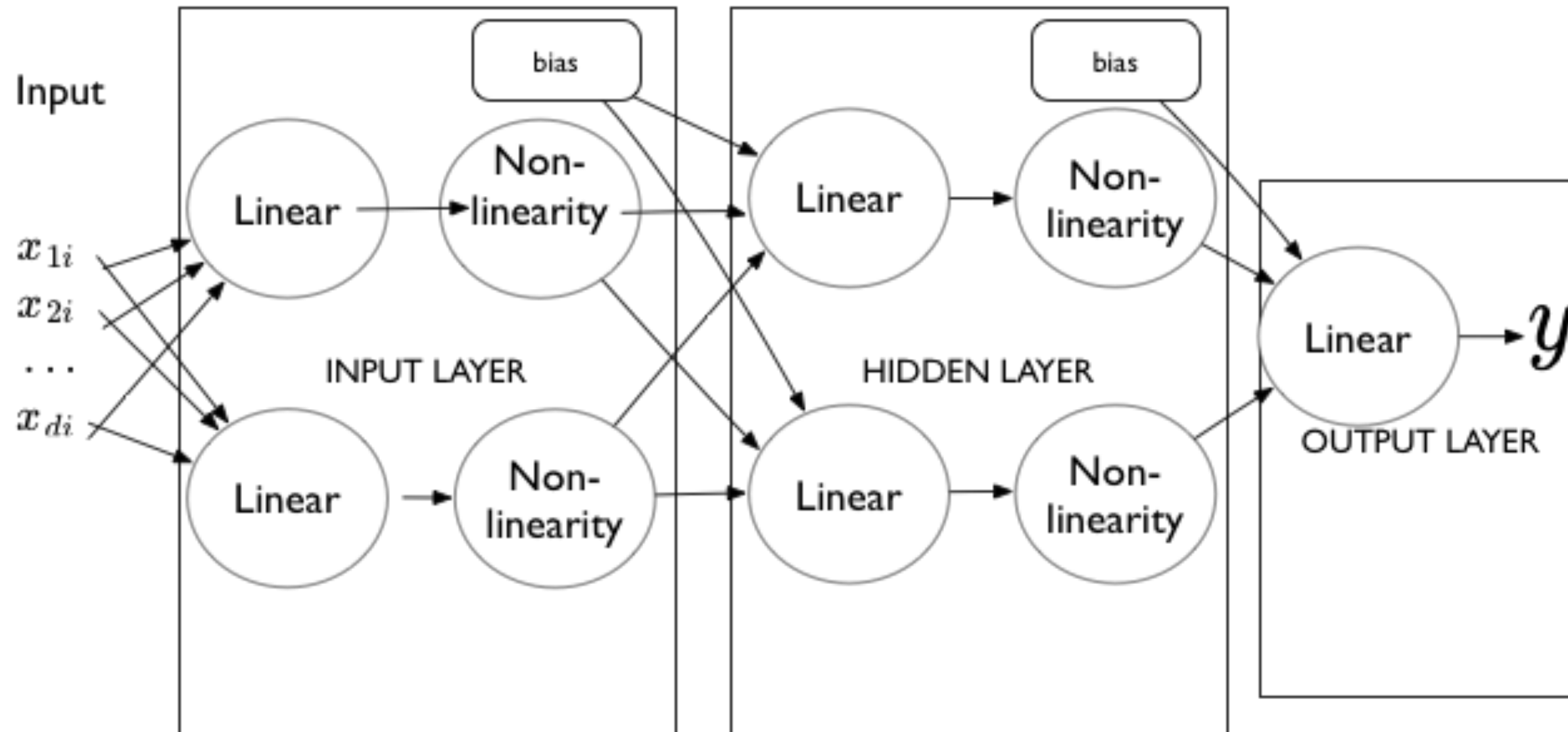
Feed Forward Neural Nets: The perceptron



Just combine perceptrons

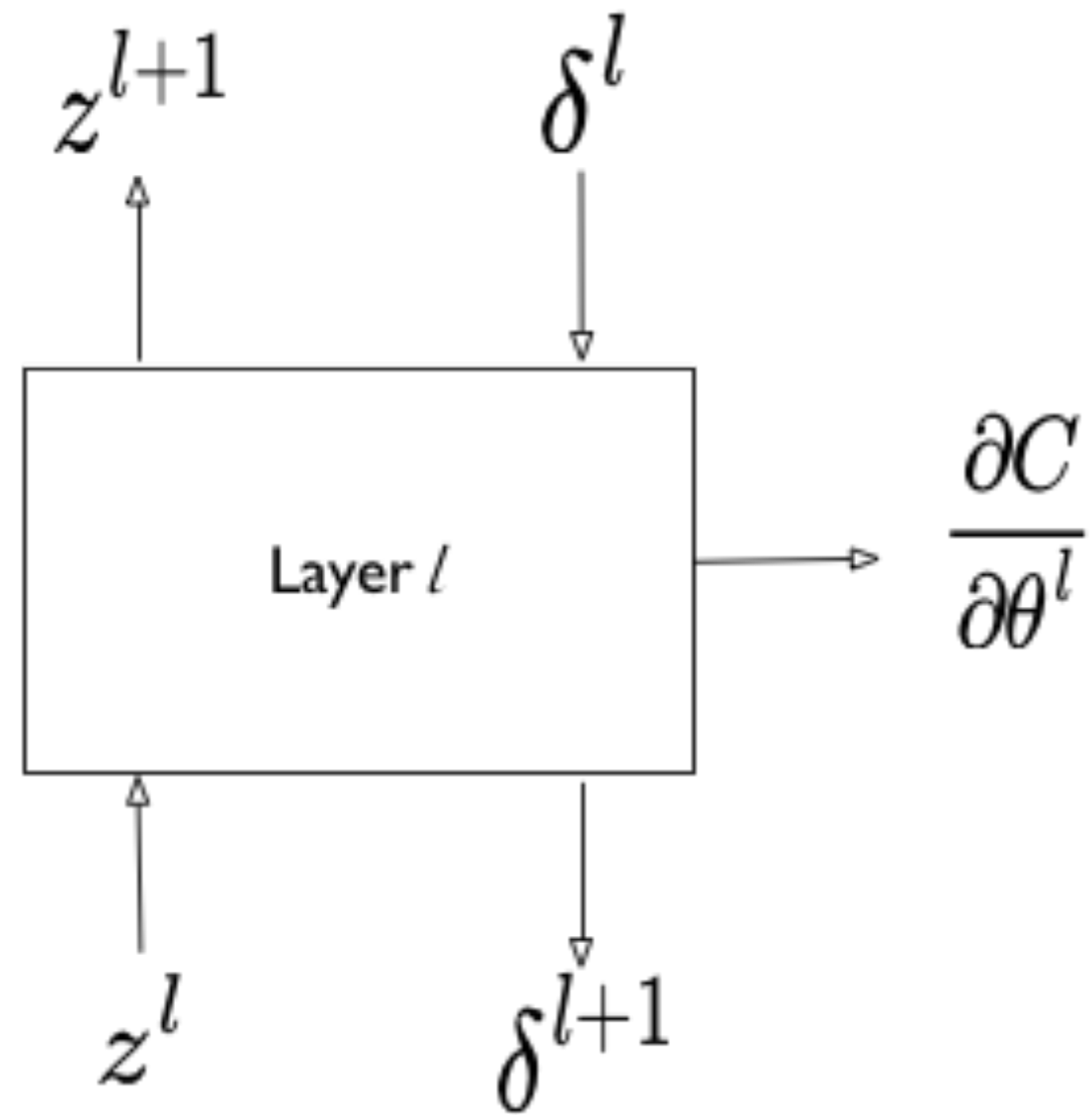
- both deep and wide
- this buys us complex nonlinearity
- both for regression and classification
- key technical advance: BackPropagation with
- autodiff
- key technical advance: gpu

Combine Perceptrons



Layer Diagram

THATS IT! Write your Own Layer



What it looks like?

See <https://github.com/joelgrus/joelnet>

Look at the video. A full deep learning library in 35 minutes!

Universal Approximation

- any one hidden layer net can approximate any continuous function with finite support, with appropriate choice of nonlinearity
- under appropriate conditions, all of sigmoid, tanh, RELU can work
- but may need lots of units
- and will learn the function it thinks the data has, not what you think